NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

DESIGN OF A FINANCIAL MANAGEMENT SYSTEM FOR THE ACADEMIC DEPARTMENTS AT THE NAVAL POSTGRADUATE SCHOOL

by

Alan E. Pires

March, 1997

Thesis Advisor: C. Thomas Wu

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March, 1997	Master's Thesis

4. TITLE AND SUBTITLE DESIGN OF A FINANCIAL MANAGEMENT SYSTEM FOR THE ACADEMIC DEPARTMENTS AT THE NAVAL POSTGRADUATE SCHOOL

5. FUNDING NUMBERS

6. AUTHOR(S) Alan E. Pires

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)*

This thesis examines the requirements and design of a financial management system for the academic departments at the Naval Postgraduate School. Existing systems are difficult to maintain and/or provide out-of-date information. A system is needed that is easy to use, easy to maintain, and provides current account status information so that the academic departments can make intelligent financial decisions.

We examined existing methods and tools for designing and building client/server applications. After comparing the traditional waterfall approach to the rapid prototyping approach, we elected to use rapid prototyping in order to develop the system quickly and to help the users determine their own requirements. We decided to use the *Powersoft Portfolio* tool set from Powersoft Corporation because it is scalable, transportable, affordable, and compliant with the Open Database Connectivity standard.

The result of this thesis is a prototype financial management system that users have found easy to use and maintain. The system provides summary and detail information on departmental financial accounts, to include balances and expenditures in the funding categories of faculty and support labor, equipment, travel, and contracts.

14. SUBJECT TERMS Accounting System, Database, Client/Server Application, Rapid Prototyping,			15. NUMBER OF PAGES 177 16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18 298-102

Approved for public release; distribution is unlimited.

DESIGN OF A FINANCIAL MANAGEMENT SYSTEM FOR THE ACADEMIC DEPARTMENTS AT THE NAVAL POSTGRADUATE SCHOOL

Alan E. Pires
B.S., United States Military Academy, 1980

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
March 1997

Author: Alan E. Pires
Alan E. Pires

Approved by:
C. Thomas Wu, Thesis Advisor

John A. Daley, Second Reader

Ted Lewis, Chairman
Department of Computer Science

ABSTRACT

This thesis examines the requirements and design of a financial management system for the academic departments at the Naval Postgraduate School. Existing systems are difficult to maintain and/or provide out-of-date information. A system is needed that is easy to use, easy to maintain, and provides current account status information so that the academic departments can make intelligent financial decisions.

We examined existing methods and tools for designing and building client/server applications. After comparing the traditional waterfall approach to the rapid prototyping approach, we elected to use rapid prototyping in order to develop the system quickly and to help the users determine their own requirements. We decided to use the *Powersoft Portfolio* tool set from Powersoft Corporation because it is scalable, transportable, affordable, and compliant with the Open Database Connectivity standard.

The result of this thesis is a prototype financial management system that users have found easy to use and maintain. The system provides summary and detail information on departmental financial accounts, to include balances and expenditures in the funding categories of faculty and support labor, equipment, travel, and contracts.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS

I would like to thank Dr. C. Thomas Wu for his continual support, guidance, patience, and sense of humor during this project. I would also like to thank LCDR John A. Daley for his help and support in editing.

I also wish to thank the Chairman of the Operations Research Department, CAPT Frank Petho, and my supervisor, Alan Jones, for their support. Without their support, this work would not have been possible. I also wish to express my gratitude to the staff of the Operations Research Department who tested the system and provided invaluable input for its design and development and to my co-workers who provided encouragement and support.

I am very grateful to my parents for their support and faith in me. Their positive attitude and encouragement have always been a source of inspiration. Most importantly, I am indebted to my wife, Kristine, and my daughters, Jessica and Monica, for their constant love, patience, and understanding during the work involved in this thesis project.

I. INTRODUCTION

A. BACKGROUND

The academic departments of the Naval Postgraduate School need a method to provide current status information for their numerous financial accounts. Reports from the Comptroller are quarterly and are frequently out-of-date when received. Without up-to-date information, the departments cannot make intelligent financial decisions. Although solutions to this problem have been developed, they do not provide a complete or efficient solution to the problem. This thesis determines the requirements and design for a financial management system for the academic departments.

B. REVIEW OF EXISTING SYSTEMS

1. Operations Research Department System

The Operations Research Department has a system that was developed using Borland Paradox for DOS. It was loosely based on a system that had been developed for the Administrative Science Department (now known as the Systems Management Department) using dBase IV [Ref. 1, 2, 3, 4, 5]. Neither the Administrative Science Department's database nor the Operations Research Department's database was designed using proper database design techniques, i.e., no data modeling was done such as through the use of Entity-Relationship (ER) diagrams or Enhanced Entity-Relationship (EER) diagrams [Ref. 6]. The Administrative Science Department's system was not easy to maintain and not easily transportable to other departments.

The Operations Research Department's system, named the "Paradox-based Financial Management Information System (PFMIS), allowed the inputting of account, labor, equipment, and travel information but only calculated the balance of accounts for the labor category. The version of Paradox used does not support storage of embedded code, such as Structured Query Language (SQL) code, in the database. Instead, scripts written in the "Paradox Application Language" have to be manually executed to perform calculations such as those needed to determine the balance of an account. More sophisticated databases allow embedded code, known as triggers and stored procedures, which can cause calculations or other actions to happen automatically upon insertion, modification, or deletion of data in the database.

2. Computer Science Department System

The Computer Science Department system is based on the Microsoft Excel spreadsheet. As such, it does not have many of the important features of a database system. For example, it cannot check that the user is inputting valid data, it cannot provide various levels of security to the data such as allowing some users read-only access and other users read-write access, it cannot provide transaction tracking and the ability to cancel transactions, it cannot provide the necessary protection to data that would allow simultaneous inputting of data by multiple users, and it cannot easily provide on-line access to individual professors of the status of their accounts. To provide account status information to the professors, the individual who inputs the data into Excel runs a program that converts a spreadsheet containing summary status information into a HyperText Markup Language (HTML) document. The HTML document is then posted on a World Wide Web page where the

professor can view it. A database system, on the other hand, would allow the professors to access the database at any time to view the status of an account or the database system could be set to automatically update a Web page whenever new data was entered. In short, the Computer Science Department is attempting to solve a database problem using a spreadsheet.

This thesis uses an approach that will use modern design techniques to provide a robust financial accounting system that is easy to use and maintain.

II. SYSTEM REQUIREMENTS AND DESIGN

A. PROJECT SCHEDULE

The first step in the project was to develop a project schedule. A copy of the schedule is given in Appendix A. The project was divided into three main phases: a design phase, a development phase, and a test/debug phase. Each of these phases consisted of a variety of tasks. It was determined that many of the tasks could be done in parallel. To begin the project, system requirements were determined and software tools were selected. The Operations Research Department was selected as the test department for the project.

B. SYSTEM REQUIREMENTS

System requirements were developed by studying the existing system in the Operations Research Department and by conducting interviews with key personnel in that department to determine what tasks they needed to perform [Ref. 7]. The system requirements were determined to be as follows.

1. General Requirements

- Track the department's financial accounts. All type of accounts need to be tracked, e.g., Reimbursable Research (RR), Direct Research (DR), Direct Teach (DT), etc.

- Track the total dollar amount of each account, as well as the subcategories that the funds are broken out to, i.e., faculty labor, support labor, travel, OPTAR, and contracts.

- Data must be exportable, i.e, the user[1] must be able to bring data from the system into a spreadsheet or other program for manipulation.

- Security down to the "field" level so that only authorized users can read and/or write fields, records, and tables.

- The "front end" of the system must be compatible with Windows 3.1x, Windows 95, Mac OS, and common variations of the Unix operating system, such as Sun Solaris.

2. Read Access (Queries)

- Determine the balance in an account broken out into the following subcategories: faculty labor, support labor, travel, OPTAR, and contracts.

- List all charges against an account and see which charges are obligations (funds committed but not spent) versus actual expenditures.

3. Write Access (Updates)

- Write access (updates) must be limited to authorized users in the department to help ensure the accuracy of the database.

- Authorized users should be able to enter information about initial funds in an account and charges against accounts. Charges against accounts will be in the subcategories of faculty labor, support labor, travel, OPTAR, and contracts. If possible, this information should come from other systems, e.g., SACONS (Standard Automated Contracting System), to avoid duplicate entry of data.

---

[1]For these requirements, the term "user" refers to any authorized user of the system, e.g., a staff member who inputs data, the department chairman, and faculty members who are the Principle Investigators for accounts.

4. Report Generation

- The user should be able to produce the faculty and staff labor certification reports for each pay period. These reports show the number of hours of labor each week charged to specific accounts for each employee. The system should include some calendar functions so that it will automatically account for holidays, etc.

- The system must have the ability to easily produce custom reports such as lists of accounts and employees, lists of expenditures on accounts, and so on.

C. SELECTION OF SOFTWARE TOOLS

At the same time that the requirements were being developed, software tools to aid in the design of the database and the development of the application were examined. The desired features of the tools were:

- Affordable

- Scalable

- An established product. By purchasing an established product, it would more likely have support available through a variety of sources to include user groups and third-party books.

- Ease of use. The tools needed to be relatively easy to learn to use.

- Require a minimum of coding. By minimizing coding the resulting system would be easier to maintain.

- Transportable. In other words, able to implement on an IBM-compatible PC, Macintosh, or Unix-based system.

- Compliant with the ODBC (Open Database Connectivity) standard developed by Microsoft. Compliance with this standard would allow the application to interface with any ODBC compliant database such as Oracle or Sybase SQL Server. This would prevent the design from being locked in on one product/vendor for implementation.

The products that were considered included: *Powersoft Portfolio*, *Symantec Enterprise Developer*, *Oracle Database Server* and *Oracle Power Objects*, and *Borland Delphi*. The decision was made to select *Powersoft Portfolio* because it provided a database design tool (*S-Designor AppModeler*, formerly, *StarDesignor*), an application development tool (*PowerBuilder Desktop*), and a database server (*Sybase SQL Anywhere*, formerly, *Watcom SQL Server*), it met all of the desired features, and it was the most affordable.

D. DATABASE DESIGN

1. The Enhanced Entity Relationship Diagram

After the system requirements had been determined, the database was designed using an Enhanced Entity-Relationship (EER) diagram [Ref. 6]. The EER diagram, minus the attributes, is shown in Figure 1. The attributes for each entity and relationship are shown in Tables 1 and 2 respectively. The EER diagram was developed based on the system requirements, interviews with users of the system, and desired reports (output) from the system. The completed EER diagram was used to determine what tables to create, what attributes to have in each table, and what relationship existed between tables [Ref. 6].

2. The Physical Data Model

The database design tool included with *Powersoft Portfolio*, *S-Designor AppModeler*, could not be used to create EER diagrams. Instead, the user graphically creates database tables, enters the attributes for each table, and then creates the relationships between tables. This is what *S-Designor AppModeler* refers to as the "physical data model." Once the physical data model is complete, the user can generate any number of ODBC compliant databases, such as Oracle, Sybase SQL Anywhere, Microsoft Access, Borland Paradox, etc.

For this project, once the physical data model had been created from the EER diagram, the physical data model was used as the design for the database. In other words, as the design was changed over time, the physical data model was updated, not the EER diagram. This was done for practical reasons. Changes could easily be made to the physical data model using *S-Designor AppModeler*. No tool was available to easily change the EER diagram. After making changes to the physical data model, the database could be modified automatically using *S-Designor AppModeler* to generate and execute the SQL code. Making changes to the EER diagram could not, of course, be used to change the database automatically since *S-Designor AppModeler* could not work with the EER diagram. The physical data model is shown in Figure 2.

The user of *S-Designor AppModeler* does have to provide some of the intelligence for modifying the database, i.e., *S-Designor AppModeler* cannot successfully implement all modifications to the database. If multiple changes need to be made to the database, the user might have to enter one change at a time to the physical data model and have *S-Designor AppModeler* modify the database after each change to the physical data model in order to have the changes implemented properly. This is not always the case. It depends on what changes are being made to the database. For example, if non-key attributes (fields) are being added to some of the tables, this could be done all at once. If, however, a key attribute was being added or removed from a table along with other changes to the same table, the changes would have to be done individually.

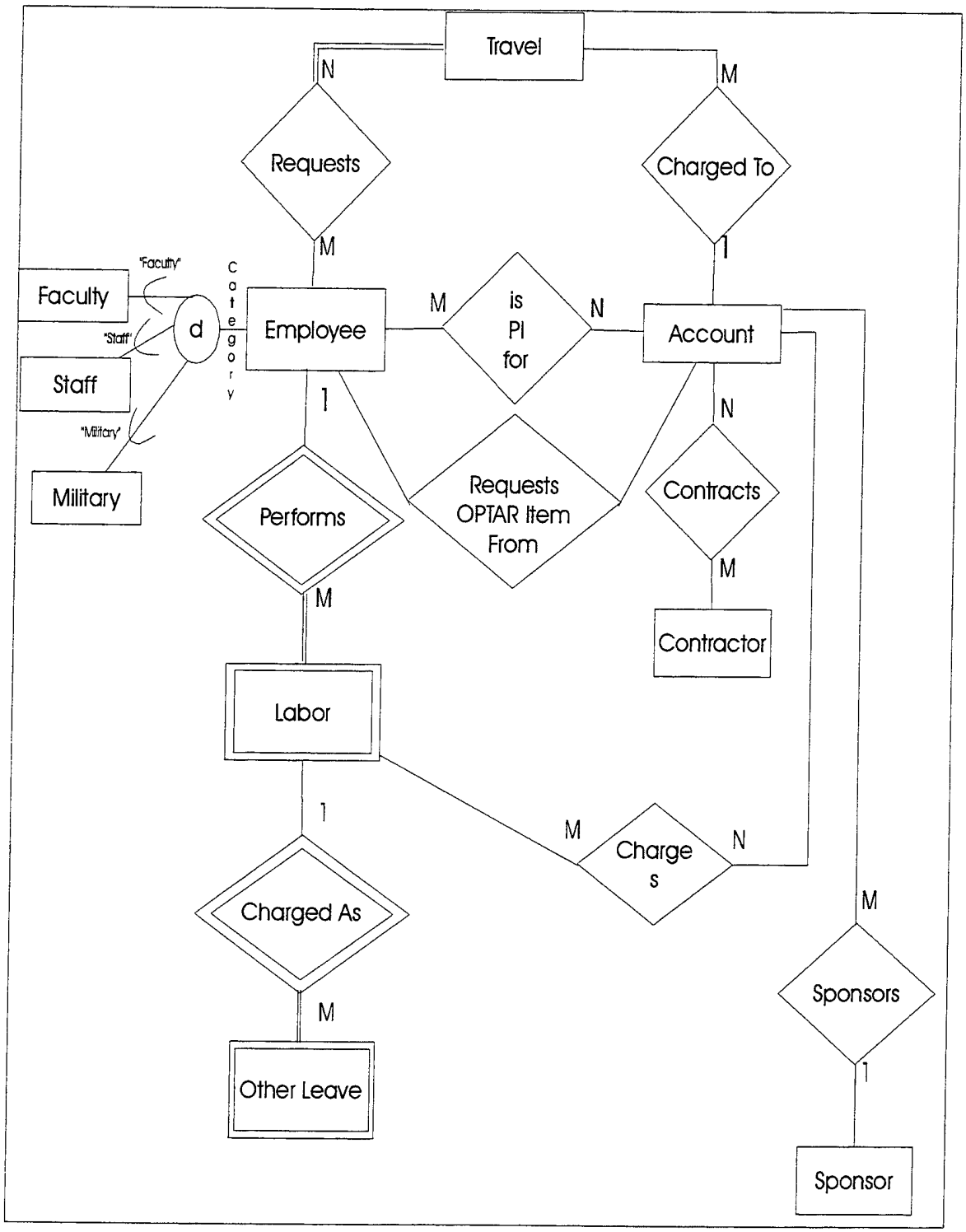


Figure 1. Enhanced Entity Relationship Diagram (Minus Attributes)

Employee	Account	Travel	Labor	Sponsor	Other Leave	Contractor
Employee ID Code	JON	TO #	PPE Date	Name	Type	Name
SSN	Budget Page Date	TO Date	A.L. Hours	Address	Num Hrs	Address
First Name	Fund Type	Proj Cost	Holiday Hrs	Phone		Phone
MI	Labor JON	Actual Cost	S.L. Hours			
Last Name	MIPR #	Trav Start Date	L.W.O.P. Hrs			
Base Salary	Title					
Accel Rate	Serial #1					
Bldg #	Serial #2					
Room #	Date Recvd					
Work Phone	Expir Date					
Home Phone	Init Fac Labor $					
Street Addr	Init Spt Labor $					
City	Init OPTAR $					
State	Init Travel $					
Zip	Init Contract $					
Categories of Employee:						
Faculty	**Staff**	**Military**				
Civ Grade	Civ Grade	Mil Grade				
Step	Step	Service				

Table 1. Attributes of Entities.

Requests OPTAR Item From	Charges	Contracts
Doc #	Hours	PO #
PO #	Overtime Hours	Proj Cost
Proj Cost		Actual Cost
Actual Cost		
Description		
PO Date		
Date Recvd		
Order Date		
Category		
ADP Proj #		

Table 2. Attributes of Relationships.

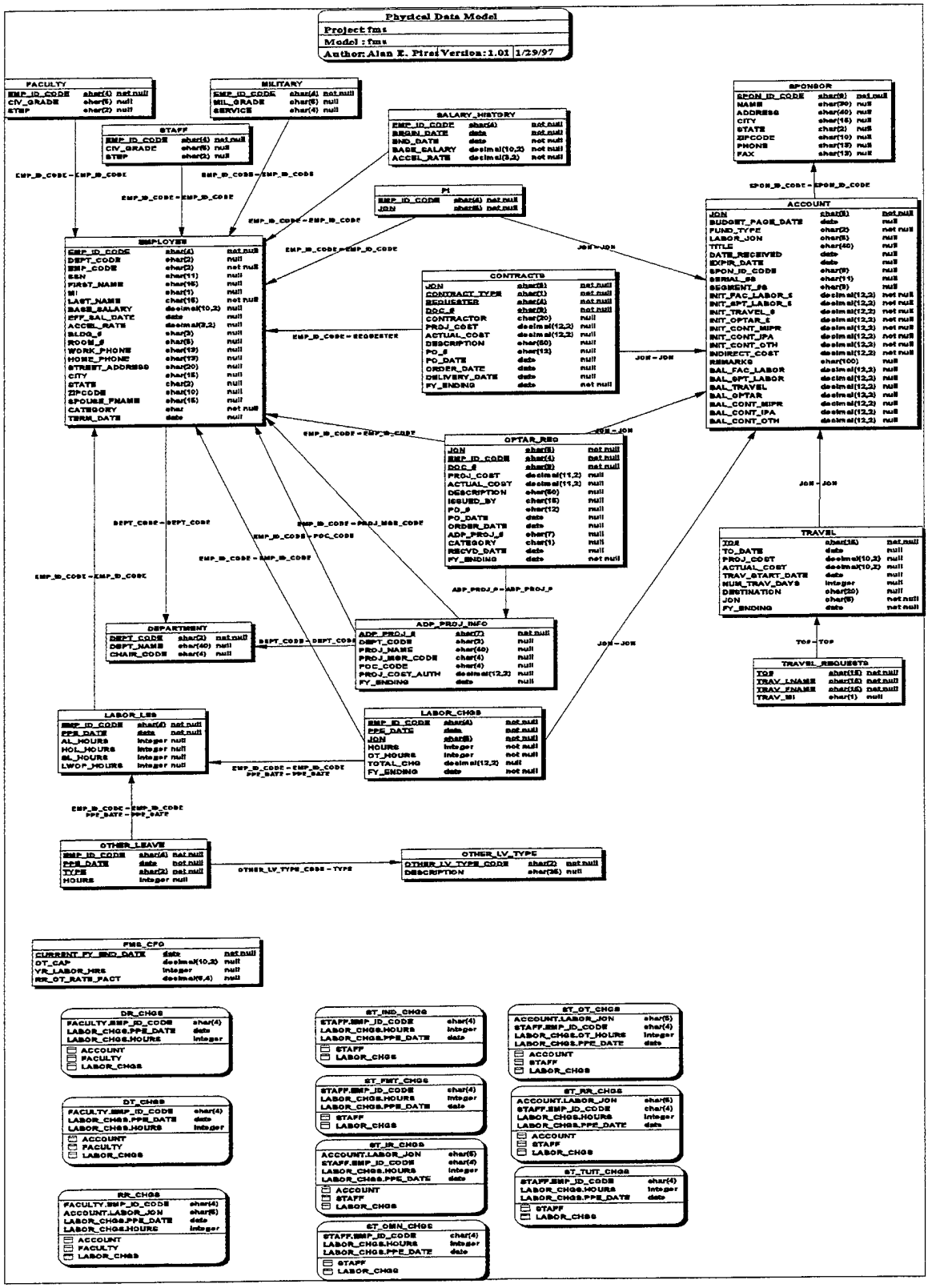


Figure 2. Physical Data Model

III. FINANCIAL MANAGEMENT SYSTEM

A. CLIENT/SERVER PROCESSING DECISION

We (my thesis advisor and I) decided to call the system the "Financial Management System" (FMS). Once the design of the Financial Management System database was complete, the development phase began. The solution being implementing utilized the "client/server" model of computing [Ref. 8] where some of the computing (processing) is done by the database residing on a "server" (a PC running the database server, in our case) and some of the computing is done by the application which runs on the "client" machine (again a PC in our case). A key part of the development phase was determining what would be done by the database ("back-end"), and what would be done by the application ("front-end").

1. Database (Back-end) Processing

The database (back-end) handles the referential integrity constraints using triggers and it handles the calculation of the balance of the accounts using stored procedures. The reason for handling the referential integrity constraints using triggers is that *S-Designor AppModeler* automatically generated most of the triggers to enforce referential integrity thus having the tool do most of the work and making the database easier to maintain. The reason for calculating the balance of the accounts using stored procedures is so that the procedure would have to be written only once. It can be called by any trigger that would affect the balance of an account. Otherwise the code to calculate the balance of an account would have had to be

placed in every trigger that affects the balance of an account. A listing of the triggers is given in Appendix B, and a listing of the stored procedures is given in Appendix C.

Handling "referential integrity constraints" refers to ensuring the consistency of the data. In a relational database, a parent-child relationship can exist between tables. With a parent-child relationship, one or more records in the "child" table can refer to a record in the "parent" table. For example, in the FMS database there is a "parent" table called "DEPARTMENT" that contains information about academic departments such as the department code, department name, etc. A "child" table of DEPARTMENT is the table called "EMPLOYEE" which contains information about employees to include the department code of the department they belong to. The referential integrity constraint triggers in a database ensure that, for example, a record in the DEPARTMENT table cannot be deleted if EMPLOYEE records still exist with that department code (i.e., there are one or more records in the "child" EMPLOYEE table which reference the record to be deleted in the "parent" DEPARTMENT table). Figure 3 shows the attributes of the EMPLOYEE and DEPARTMENT tables and the arrow in the Figure from the attribute DEPT_CODE in the EMPLOYEE table to the attribute by the same name in the DEPARTMENT table illustrates the reference.

These integrity constraint "triggers" are Structured Query Language (SQL) code [Ref. 6] that are automatically executed upon occurrence of an event. The events that cause triggers to executer ("fire") are inserting, updating, and deleting of records. Triggers can be set to occur either before or after each of these events. *S-Designor AppModeler* automatically creates integrity constraints triggers for tables that have parent-child

relationships. The tasks performed by the triggers automatically created by *S-Designor AppModeler* include:

- The insert triggers ensure that a "parent" record exists (in the parent table) for every record inserted in a "child" table. If the parent record does not exist, the trigger does not allow the child record to be inserted.

- If the parent-child relationship is set to "delete prohibit," delete triggers will not allow the deletion of a "parent" record if a "child" record still exists. However, if the relationship between a parent and child table has been set to "cascade" delete, the delete triggers will automatically delete child records if a parent record is deleted.

- The update triggers ensure that the field of a parent record which links it to a child record cannot be changed unless the trigger is set to automatically change the corresponding field in the child record.

The stored procedures which calculate the balance of each account are also SQL code. These stored procedures are called by triggers. When an event occurs that would change the balance of an account, such as the insertion of a travel record (i.e., a travel expense), the trigger causes the stored procedure to execute that calculates the travel balance of the account to be charged.

2. Application (Front-end) Processing

The application handles data validation. In other words, it only allows the user to enter data which meets data integrity constraints. For example, the application will not allow the user to enter a negative number for the number of days an individual was on travel. Of course the application cannot stop the user from entering incorrect data. For example, the user could enter that an individual was on travel for five days when they were actually on travel for three

days. The application would not catch the incorrect entry because five is in the range of valid numbers allowed to be entered in the field.

B. APPLICATION DEVELOPMENT

1. Background

As stated previously, a product called *PowerBuilder Desktop* was used to develop the application (front-end) of the FMS. *PowerBuilder* is a graphical application development tool for developing client/server applications that access databases. *PowerBuilder* provides pre-made standard window controls such as buttons, radiobuttons, checkboxes, dropdown listboxes, etc., to minimize the amount of coding that needs to be done by the developer. It also provides a scripting language with built-in functions which also help to minimize coding. Typically scripts are executed when an event occurs such as when a user clicks on a button.

A *PowerBuilder* application is made up of objects such as windows and menus. Objects are stored in *PowerBuilder* libraries and retrieved from these libraries when the application is run. Some of the types of *PowerBuilder* objects are:

- Application Object: the entry point into an application which defines application-level behavior such as what the default text font is and what processing should be done when the application begins or ends.
- Window Objects: the interface between the application and the user. They request information and display information.
- DataWindow Objects: used for retrieving and manipulating data from a relational database or other source such as a spreadsheet. It also determines the style of presentation of data such as tabular or freeform. Output from the database such as reports are retrieved and displayed using DataWindow objects.
- Menus: provides the user of the application with a list of choices (actions) to select from such as listing reports that can be produced.

- Global Functions: independent objects that perform general-purpose processing such as string handling.

- Queries: a SQL statement that is used to retrieve data from a relational database and saved with a name so that it can be reused. Normally they provide data for a DataWindow object.

- Structures: a collection of one or more related variables of possibly different data types grouped under a single name. This corresponds to the data structure called a "record" in Pascal and other programming languages. Structures allow the developer to refer to a set of related items as a single unit, rather than having to refer to multiple items.

- User Objects: an application feature defined by the user so that it can be reused in one or more applications.

- Libraries: as stated previously, *PowerBuilder* libraries are used to store objects. Applications retrieve the objects from the libraries so libraries can be shared by multiple applications.

- Projects: packages the application for execution by the application user(s). The application can be packaged as a stand-alone executable or as an executable that links to *PowerBuilder* dynamic libraries at execution time.

2. Implementation

a. Financial Management System Modules

The FMS, when complete, will consist of three modules (projects, in *PowerBuilder* terminology) -- a staff module, a faculty module, and a chairman module. The purpose of the staff module is to provide the means for the academic department's administrative staff to input data into the system and produce reports. The purpose of the faculty module is provide the means for the academic department's faculty to check the status of the research accounts for which they are assigned as the principal investigator. The purpose of the chairman module is provide the means for the academic department's chairman

to check the status of all of the department's accounts and to perform planning and other accounting functions unique to the department chair. The staff module was developed as the prototype system for this thesis research project. The faculty module is developed but will not be discussed in this thesis.

b. Staff Module Components

The staff module of the FMS revolves around two main components as reflected by the majority of window objects used in the module. These window objects are employee related windows and account related windows. For both employees and accounts, there are list windows for providing a listing of all records with a minimum of attributes shown, detail windows for showing all of the attributes of one record, and search windows for searching for a specified employee or account record. From the employee detail window, the user can add or modify an employee record. (Note: employee records are normally not deleted. If an individual ceases to be a Naval Postgraduate School employee for whatever reason, an employment termination date attribute is filled in. If an employee record needs to be deleted because it was added in error, the staff member who made the entry asks the database administrator to delete the record.)

A screen shot of the employee detail window is shown in Figure 4. The employee detail window shows the accounts (if any) the employee is the principal investigator for. Every research account is assigned one or more principal investigators who are responsible for overseeing the research and authorizing the expenditure of funds in the research account in support of the research. Funding for the account is broken out into the following

categories: faculty labor, support labor, OPTAR (equipment), travel, and contracts (broken out as MIPR, IPA, and other contracts).

The account detail window displays details about the account such as the expiration date of the account, the account sponsor, and the initial and current balance of the account in each of the funding categories. A screen shot of the account detail window is shown in Figure 5.

As can be seen from Figure 5, there is a tab for each general funding category of the account. By clicking on a tab, the user can display more details about expenditures in that category. Example screen shots of expenditures for the labor, OPTAR, and travel funding categories of an account are shown in Figures 6, 7, and 8, respectively. When the user (staff member) clicks on a funding category tab, she can then add, modify, or delete records of expenditures for that funding category of the displayed account.

The *PowerBuilder* objects used by the staff module are stored in seven *PowerBuilder* libraries. The libraries are:

- *fms_main.pbl*. This object contains the main objects for the FMS staff module such as the main menu, the main window, the password window for logging in to the system, the "about" window which gives version and authorship information about FMS, and the toolbar configuration window which allows the user to select where to place the toolbar (sometimes known as a buttonbar). The toolbar allows the user to readily access employee, account and other windows by clicking on the buttons on the toolbar.

- *fms_emp.pbl*. This object contains employee related objects such as the employee detail window, the employee list window, the employee search window, and an employee list DataWindow for printing a list of employees.

- *fms_acct.pbl.* This object contains account related objects such as DataWindows for labor, OPTAR, travel, and contract expenditure listings for an account. These objects are shared by the faculty module of the FMS.

- *fms_acc2.pbl.* This object contains account related objects used solely by the staff module of the FMS such as the account list window, the account detail window, and the account search window.

- *fms_mnt.pbl.* This object contains maintenance related objects such as windows and DataWindows for adding, modifying or deleting records of labor, OPTAR, travel, and contract expenditures and adding, modifying or deleting records of sponsors of research accounts. These objects are shared by the faculty module of the FMS.

- *fms_mnt2.pbl.* This object contains maintenance related objects used solely by the staff module of the FMS such as windows and DataWindows for adding, modifying, and deleting employee and account records.

- *fms_rpt.pbl.* This object contains report related objects such as DataWindows for producing reports on labor, OPTAR, travel, and contract expenditures.

A complete listing of the objects contained in each *PowerBuilder* library of the FMS staff module is in Appendix D.

c. Rapid Application Development

A methodology that was used in developing the FMS staff module is known as Rapid Application Development (RAD) [Ref. 9]. This methodology, also known as 'Rapid Prototyping,' seeks to speed the development of a system by developing a quick prototype of the system, demonstrating the prototype to the eventual users of the system for their input, making changes to the system based on the users input, and repeating the cycle until a deliverable product is developed [Ref. 10, 11]. As we developed the FMS staff module, we demonstrated it every two to four weeks to the Operations Research Department staff members who would be using the system. At times, the staff input not only resulted in

changes to the design of the application but also to the design of the database. Fortunately, the tools we were using, *S-Designor AppModeler* and *PowerBuilder Desktop*, allowed us to make changes to the database design relatively easily and with minimal impact on the application.

C. APPLICATION DEPLOYMENT

Once the FMS staff module prototype was developed to the point of being usable and with no obvious bugs, it was installed in the Operations Research Department for testing and debugging. Staff members were given a brief instruction on how to use the system and asked to use the system in parallel with existing systems to check the accuracy of the FMS. Staff members were also asked to report in writing all bugs they discovered and to request desired enhancements to the system in writing. Bug reports were evaluated to determine if an actual bug existed or whether the problem was due to operator error. If an actual bug existed, it was fixed and the fix was installed as soon as possible. Enhancement requests were evaluated to determine if they could reasonably be implemented. If so, the enhancement was made and installed. If not, the requester was notified why the requested enhancement could not be made to the system.

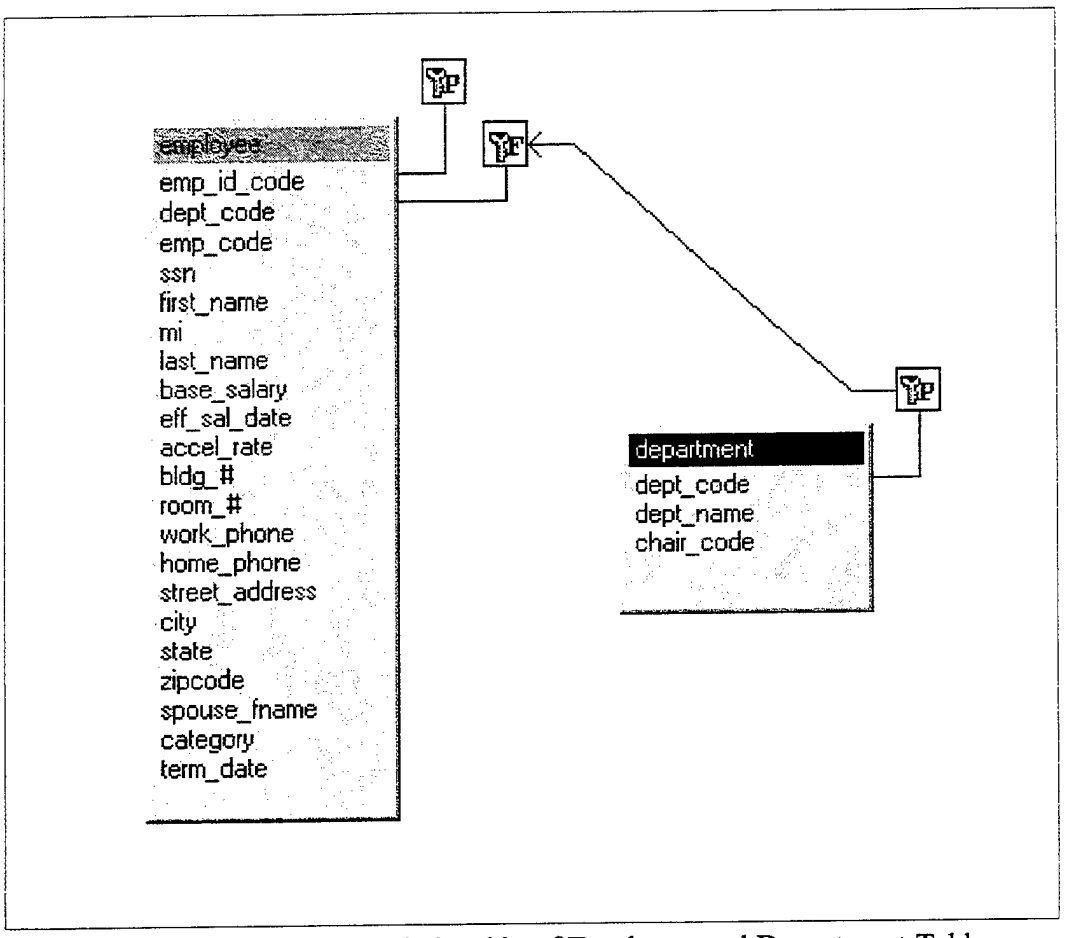


Figure 3. Parent-child Relationship of Employee and Department Tables

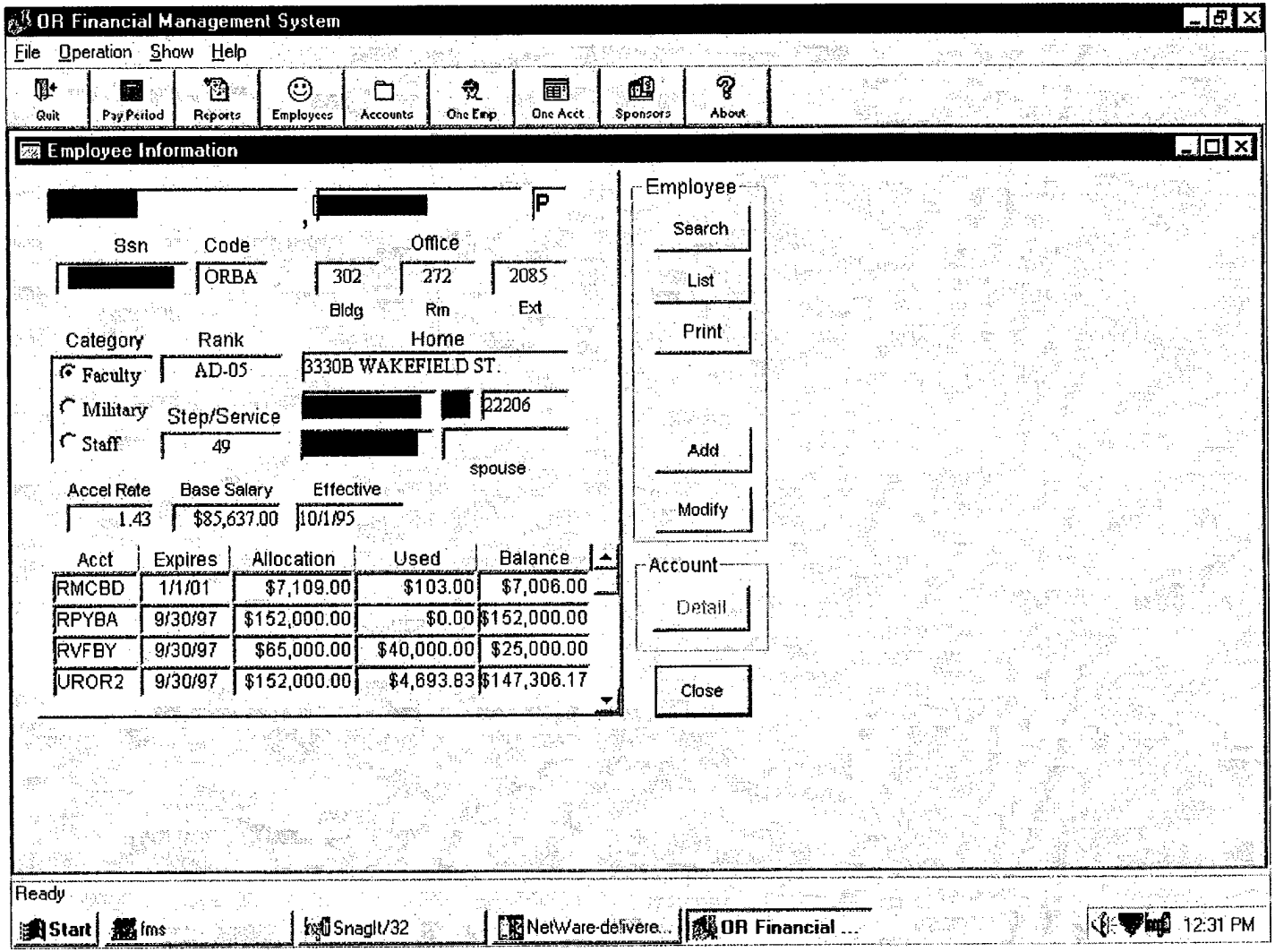


Figure 4. Employee Detail Window

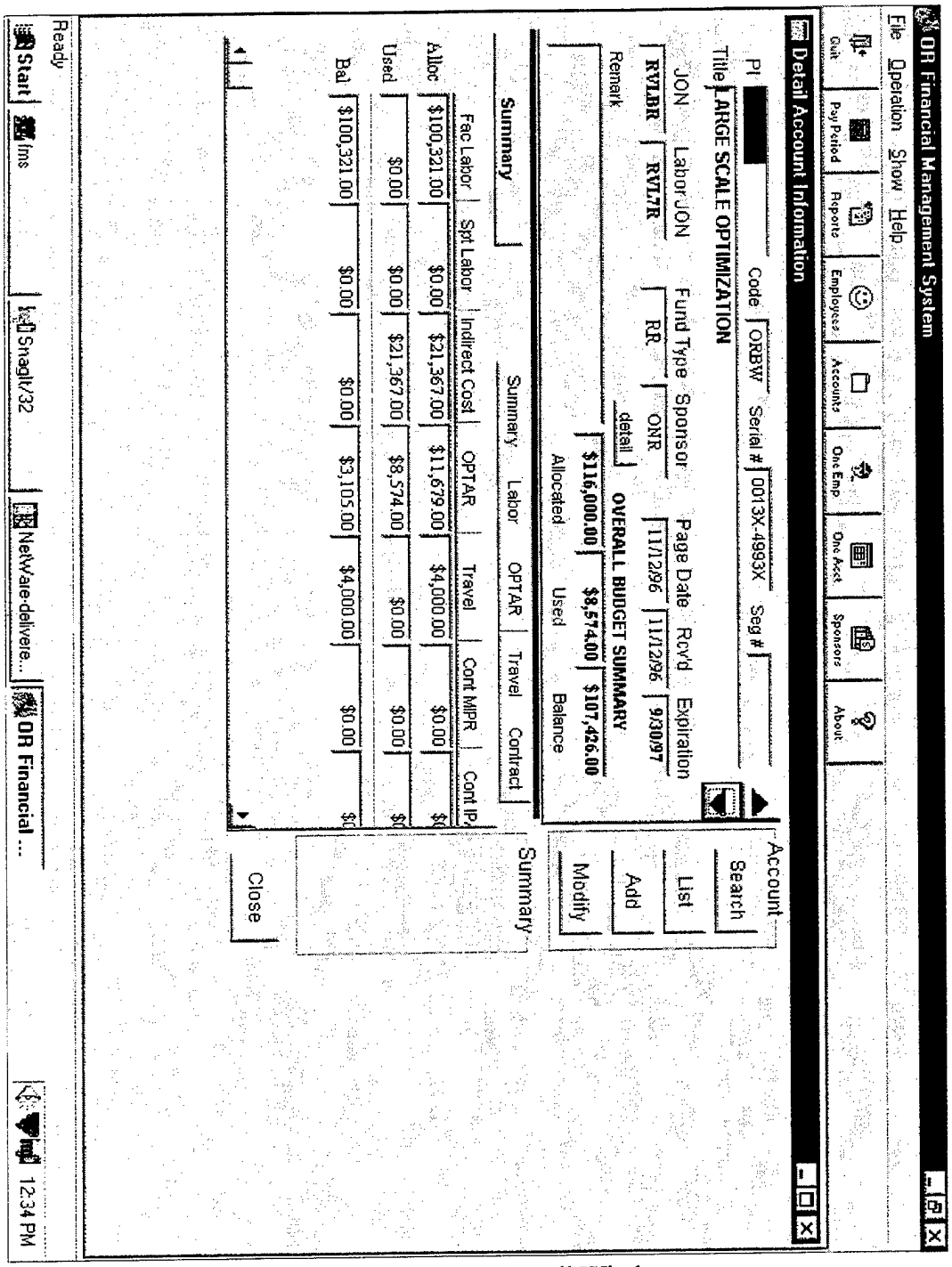


Figure 5. Account Detail Window

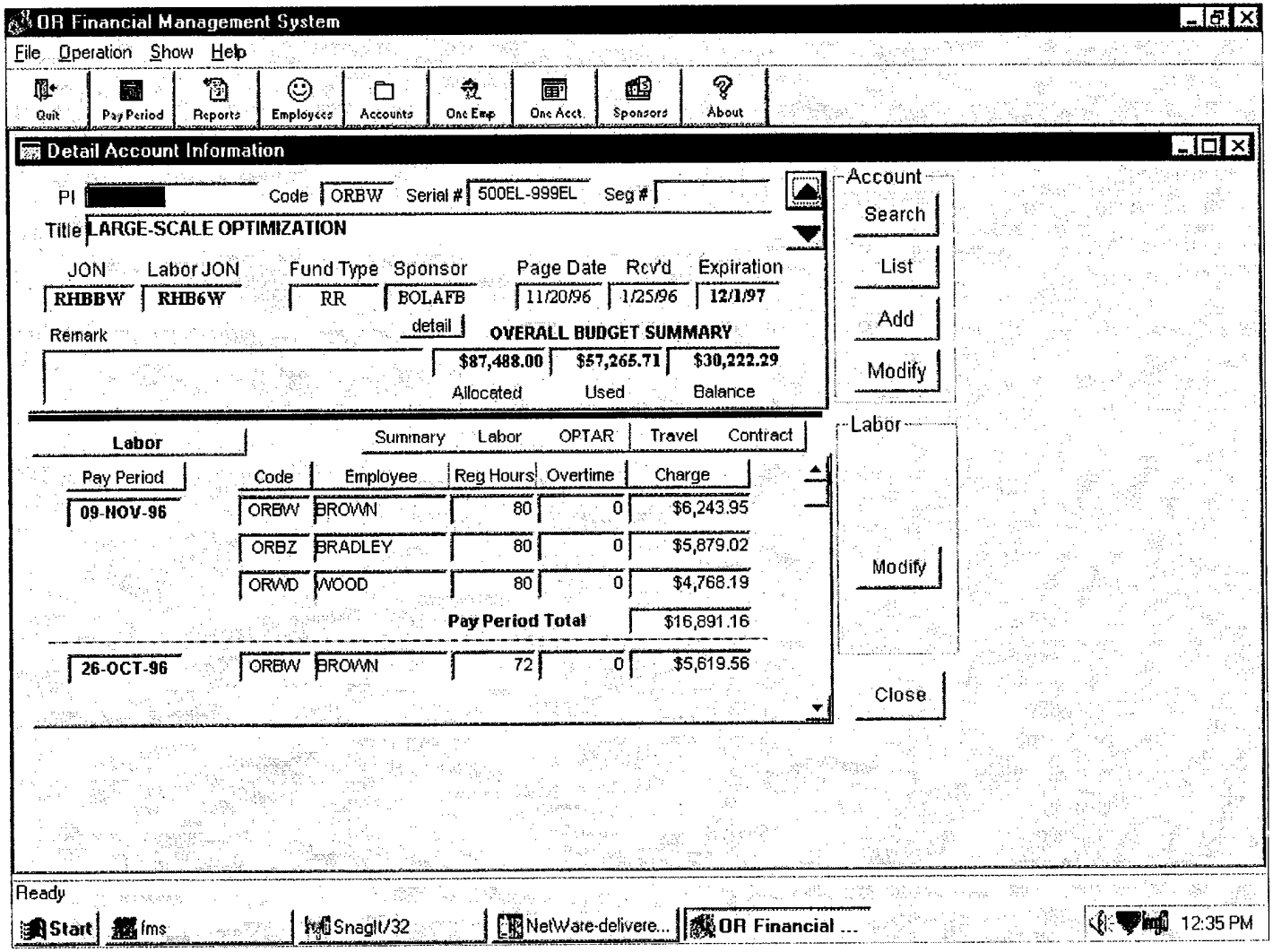


Figure 6. Account Detail Window Showing Labor Expenses

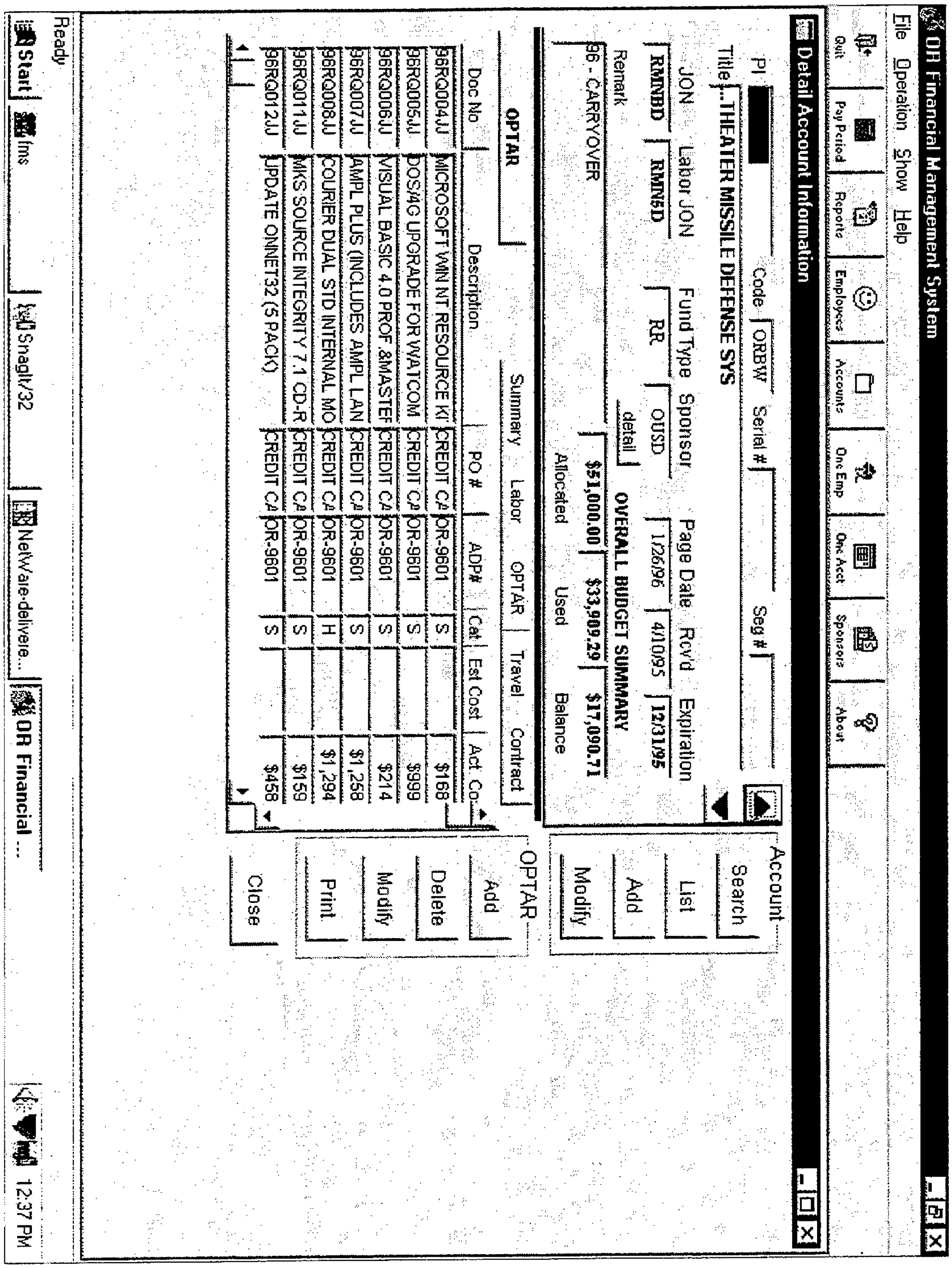


Figure 7. Account Detail Window Showing OPTAR Expenses

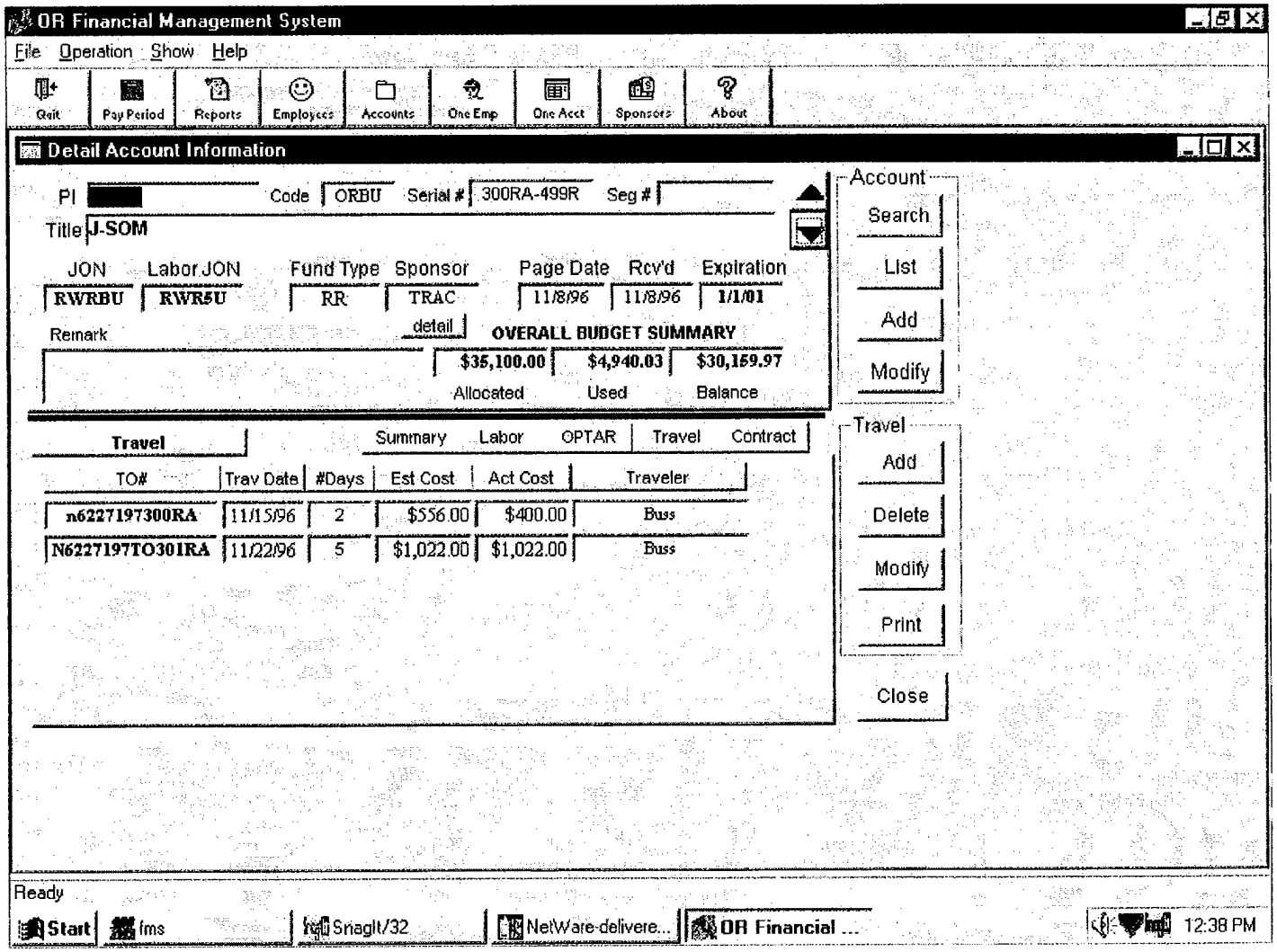


Figure 8. Account Detail Window Showing Travel Expenses

IV. ANALYSIS

A. TOOLS

1. Database Modeling

The database modeling tool used, *S-Designor AppModeler* from Powersoft Corporation, allows the user to create a graphical representation of some of the components of a relational database. This includes tables, table attributes, relationships between tables, and views. These components are stored in what *S-Designor AppModeler* refers to as the "physical data model." Other components of the relational database, such as indexes, triggers, and stored procedures, can be created as part of the physical data model using *S-Designor AppModeler* but are not shown in the graphical representation.

Overall, we found *S-Designor AppModeler* (hereafter referred to as *AppModeler*) to be a very useful database modeling tool. As with any software tool, it has its strong points and weak points.

a. Strong Points

- Overall ease of use. The user interface is fairly simple and straightforward. We were able to start using it with only a minimal amount of reading of the *User's Guide* and the on-line help. Sample physical data models were provided which also helped with learning how to use *AppModeler*. For preparing the graphical portion of the physical data model, several *AppModeler* tools are available in a tool palette: a table tool, a reference tool (for indicating the relationship between tables), a view tool, and so on. These tools in the tool palette make it simple for the user to create the tables, relationships, and views that are part of a database. A screen shot of *AppModeler* with the tool palette and the FMS physical data model is shown in Figure 9.

- Automatic generation of the database. Once the user has completed a physical data model, with the click of the mouse, the database can be generated. The user has the option of having *AppModeler* generate the database, or generate an SQL script which can be executed separately to generate the database. Before the database or SQL script are generated, *AppModeler* automatically checks the model for correctness. The user can generate the database for any of a number of target databases such as *Sybase SQL Anywhere* and *Oracle*. Many other options are available. A screen shot of the *AppModeler* database generation screen is shown in Figure 10.

- Automatic modification of the database. Automatic modification of the database is both a strong point and a weak point (see below). To modify the database, the user archives the current (prior to the changes) physical data model, makes changes to the physical data model, and then selects the Modify Database command. The user can choose to modify all tables or specify which tables to modify, modify all indexes or specify which indexes to modify, and modify all triggers and procedures or specify which triggers and procedures to modify. As with the automatic generation of the database, the user can choose to modify the database directly or to have an SQL script generated which can be executed separately to modify the database. It was very useful to select the option to generate the SQL script to check over what *AppModeler* was going to do to modify the database. If it appeared that the script would accomplish the intended modification, then the option to directly modify the database was selected. A screen shot of the *AppModeler* database modification screen is shown in Figure 11.

- Automatic generation of indexes. Indexes provide an ordered list of the records of a table based on a key field. There are two types of key fields, primary and foreign. A primary key consists of one or more fields (attributes) that uniquely identify a record in a table. A foreign key is a field that depends on and migrates from a primary key in another table. With a few mouse clicks, the database indexes for key fields (both primary and foreign) can be automatically generated or, after modification of the database, regenerated.

- Ease of creating relationships between tables. As mentioned previously, there is a "reference" tool in the *AppModeler* tool palette for creating relationships between tables. The user clicks on the Reference tool in the tool palette, clicks on the child table and drags the reference to the parent table. If the foreign key in the child table has the same name as the primary key in the parent table, those fields are automatically selected for the relationship. The user can specify which fields to use for the relationship if the correct fields are not automatically selected.

- Automatic generation of referential integrity constraint triggers. *AppModeler* automatically created referential integrity constraint triggers for tables with parent-

child relationships. In every case, the triggers automatically generated by *AppModeler* worked correctly.

- Ease of creating and modifying triggers and stored procedures. In order to have the balance of the various funding categories of accounts calculated automatically, we had to create and modify some triggers and stored procedures. *AppModeler* made this task relatively easy by providing the means to list all triggers and procedures, listing triggers by table, and allowing the user to edit them with a simple but adequate text editor. As mentioned previously, once the user had created or modified the trigger or stored procedure, he could automatically add it to the database or modify it in the database using the automatic modification feature of *AppModeler*.

- Automatic documentation (report) generation. *AppModeler* can automatically generate three types of reports: a full report which contains all main model items, a standard report which contains physical data model graphics, and all table-dependent items, and a list report which contains a single title item and all list-type items. User-defined reports can also be created. The user can print the report or save it in "Rich Text Format" to a file. Additionally, the user can choose to print the physical data model graph in color or black and white and can have *AppModeler* automatically scale the graph so that it fits on one page (an extremely useful feature). Part of the *AppModeler* full report (database schema information) for the FMS physical data model is given in Appendix E.

b. Weak Points

- Automatic modification of the database. If too many changes were attempted at once, *AppModeler* did not have the intelligence to perform them in an order that would achieve the desired results and thus end up with a physical data model that did not match the actual database. That is why it is extremely helpful for the user to first have *AppModeler* generate the SQL script and to check the script before having *AppModeler* directly modify the database. The other problem observed was that frequently *AppModeler* could not perform modification of a key field because it did not have the intelligence to perform the necessary steps. Modifying a key field usually had to be done manually in several steps. First, the data from the table had to be exported to a comma-delimited file. Then the user had to delete any relationships with the table and the table itself and use the automatic modification feature to implement this on the database. Then the user had to recreate the table with the desired change to the key field and recreate the relationships for that table and again use the automatic modification feature to implement the changes on the database. Finally, the user had to import the data

from the comma-delimited file back into the table. On occasion the user had to first manipulate the contents of the comma-delimited file (using a spreadsheet or other program) to get it into a form that would be accepted by the modified table before importing it into the modified table. In other words, the automatic modification feature was, at times, dangerous and/or time-consuming.

- Graphical representation of the database. This was a weakness in the sense that *AppModeler* could not work with an EER diagram. A preferable method is to create and modify an EER diagram and have *AppModeler* generate the table, attributes, relationships, and so on, from that.

- Automatic generation of relationships. The automatic generation of relationships (references) in *AppModeler* created a relationship between every primary and foreign key with the same name. In our case, this created many relationships that were not intended and so we found it far easier to manually create the desired relationships using the Reference tool in the tool palette.

2. Application Development

The application development tool used was *PowerBuilder Desktop* from Powersoft Corporation. *PowerBuilder* is a tool for developing graphical client/server applications that access relational databases. As such it attempts to minimize the amount of coding done by the developer in order to make it easier and faster to develop and maintain the application.

Overall, we found that *PowerBuilder* did live up to its stated purpose of easing the development and maintenance of an application. Some of its strong and weak points are listed here.

a. Strong Points

- Pre-made standard window controls. *PowerBuilder* made it easy to design menus and other standard windowing controls and thus saved a great deal of coding.

- Ability of multiple applications to share libraries. Some of the libraries were used for multiple modules (projects) of the FMS, which made it much quicker to develop the modules and maintain them.

- Reusable objects. PowerBuilder objects we created, such as DataWindows, were saved in libraries and reused within a module (project) and by multiple modules.

- *PowerBuilder* Painters. Similar to the tool palette of *AppModeler*, *PowerBuilder* had "painters" for creating *PowerBuilder* objects such as DataWindows, Applications, Projects, Menus, and so on. These painters provided an easy to use interface for creating these objects.

- Support. *PowerBuilder* is a fairly widely used product and consequently there exists a support forum for it on the computer service called CompuServe. The support forum is available at no extra charge for CompuServe subscribers and is made up of users of *PowerBuilder* (not Powersoft employees). On the occasions where we ran into problems with *PowerBuilder* that we could not solve, we posted a message detailing the problem on the support forum on CompuServe and received an answer usually within twenty-four hours that solved the problem. This form of support was important for keeping the cost of the project down since technical support from Powersoft is not free.

b. Weak Points

- Difficulty in changing fonts and font sizes. For various reasons, the font and/or font size for some of the windows and reports were changed several times. Unfortunately there was no means available to make a global change. Consequently, each text object had to be changed individually, making it a very tedious and time consuming process.

- Scripting language awkward. The scripting language is not designed logically. Too many features are ad hoc add-ons.

- The executable is not truly compiled. It requires the application's dynamic library files in order to work.

- Inadequate documentation. The manual for *PowerBuilder* was the smallest of the manuals for the three programs that made up *Powersoft Portfolio*. Not only was it the smallest but it was also the least adequate. We found it necessary to purchase third-party books about *PowerBuilder* to supplement the manual.

B. DATABASE SERVER

The database server used is *Sybase SQL Anywhere*. *Powersoft Portfolio* included a four-user version of *Sybase SQL Anywhere*. That means that four individuals can concurrently be logged in to the database server (users accessing the FMS application are logged in to the database server). This database server, in previous releases, was know as *Watcom SQL Server*. The dialect of SQL implemented by *Sybase SQL Anywhere* is Watcom-SQL. (Note: Every database server implements its own "dialect" of SQL that consist of what might be called "standard" SQL plus some extensions to it. It is similar to the various implementations of programming languages such as Pascal, BASIC, FORTRAN, and so on, by software vendors.) The database server allows a database application to communicate with a database over a network and it handles the processing done by the database, i.e., the "back-end" processing of a client/server application. Users must enter a valid user ID and password to make a connection (log in) to the database server. The *Sybase SQL Anywhere* server will run on a variety of platforms including: Novell NetWare, Windows 95, Windows NT, OS/2, Windows 3.x, and DOS. No matter what platform that *Sybase SQL Anywhere* is running on, it can be accessed by clients operating with different operating systems, such as DOS, Windows 95, Macintosh, running on different kinds of networks such as Novell NetWare, Windows NT, and Banyan Vines.

Overall, we were pleased with the *Sybase SQL Anywhere* database server. Some of its strong and weak points are listed here.

1. Strong Points

- Runs on multiple platforms. At first we ran the database server on a Novell NetWare server. During a time period when we were having a problem with the database server, occurrence of certain events could cause the database server to crash. When trying to recover the database server from the crash, it would sometimes cause the Novell server to crash. Because *Sybase SQL Anywhere* runs on a variety of platforms, we were able to move it to run on a networked PC running Windows 95 so that if the database server crashed, it did not affect the Novell server.

- Ease of use. *Sybase SQL Anywhere* was very easy to start up and administer.

- Support. As with *PowerBuilder*, a support forum is available on CompuServe for *Sybase SQL Anywhere* that is free for CompuServe subscribers. Also as with *PowerBuilder*, we posted problems we had with *Sybase SQL Anywhere* on the forum and received correct solutions usually within twenty-four hours.

- Documentation. *Powersoft Portfolio* contained three manuals for *Sybase SQL Anywhere*. These included a Watcom-SQL reference that we made good use of for writing the stored procedures and triggers for the FMS. These manuals were also available on-line so the user can easily search for specific topics.

2. Weak Points

- No automatic backup of the database. When the database server is running, the database files are open. Software for tape backup systems cannot backup files that are open. We wanted to have regular backups of the database but that meant we had to shut down the database server at the end of the workday (the tape backup automatically ran at night) and then start it up again at the beginning of the workday. It would have been very helpful if the database server could have been automatically scheduled to start and stop at specified times.

- Database server crash caused Novell server crash. As mentioned in the strong point about *Sybase SQL Anywhere* running on multiple platforms, for a time we had a problem with the database server crashing and, in turn, causing the Novell server it was running on to crash. That was very disruptive to the users of the Novell server and was totally unsatisfactory. We did receive information via the forum on CompuServe on how to fix the problem but we decided to move the database server off the Novell server to a PC just to be safe.

- Inability to handle a query with many outer joins. The event that caused the database server to crash was the execution of a query with many outer joins. This problem was a bug that had purportedly been fixed in an earlier release of *Sybase SQL Anywhere* but had apparently been reintroduced into the version we were using. The end result was that the queries had to be rewritten without the outer joins since *Sybase SQL Anywhere* could not handle them even though it was supposed to be able to do so.

C. PROTOTYPE

The FMS prototype was installed in the Operations Research Department for testing and debugging in September 1996. As with any new system, many bugs have been discovered and a variety of enhancements have been requested but overall, we believe the system has been well received. A listing of strong and weak points follow.

1. Strong Points

- Ease of use. The users of the FMS were provided with very brief instructions on how to log in to the application and do a few simple tasks. They have been able to effectively use the system without any additional instruction.

- Maintainability. We have been able to make changes to the system to fix bugs and to implement enhancement requests with relative ease. Bugs are usually fixed within a few hours. Simple enhancement requests have also been completed within a few hours but the more complex enhancement requests (ones that involved a design change) have taken a couple of days to implement (lapsed time -- the actual work took no more than a day per added feature). The ease of maintainability is due in large part to the software tools we have been using as discussed earlier in this chapter.

2. Weak Points

- Error messages. Due to a lack of time, we have not prepared error messages for all of the situations that users can cause errors. In situations where the FMS does not trap errors and provide an error message, error messages are generated by the

Sybase SQL Anywhere database server. Probably the most frequent error the user makes is to attempt an action that violates referential integrity. The error messages produced by the database server in these (and all other) situations are not comprehensible to the ordinary user. Instead, the error messages confuse the user and discourage him from using the system. We are correcting this deficiency as time permits.

- Lack of user generated reports. We have not provided the user with a means to generate reports of his own design. The complexities involved in providing such a capability to the user dictate that if it is implemented, it will provide a fairly rudimentary report generation capability. It may be possible, however, to train the users to utilize a Powersoft product called *InfoModeler* to produce reports. One of the purposes of *InfoModeler* is to provide an easy means for end-users to produce reports from a *Sybase SQL Anywhere* database.

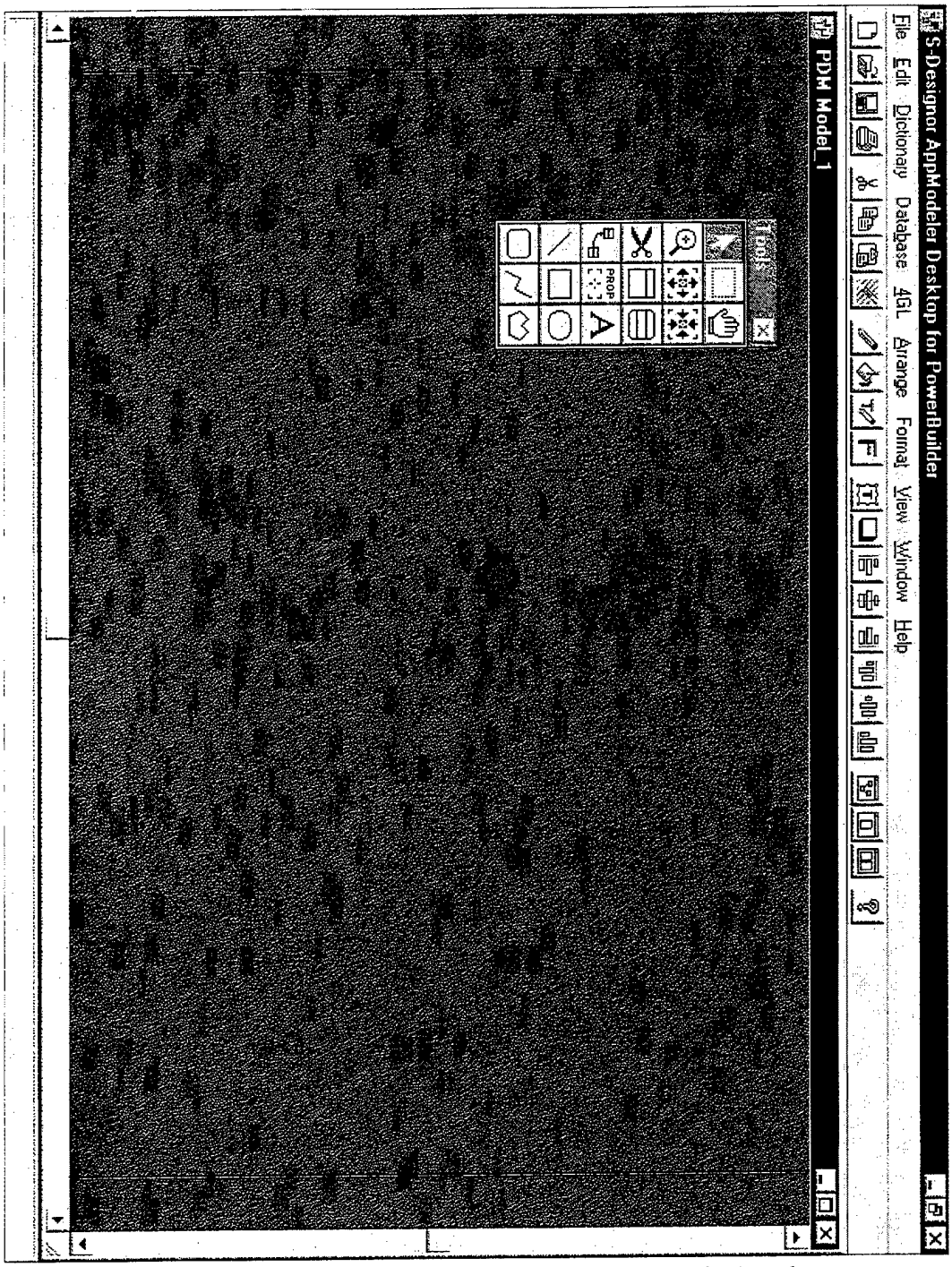


Figure 9. *AppModeler* With the Tool Palette Displayed

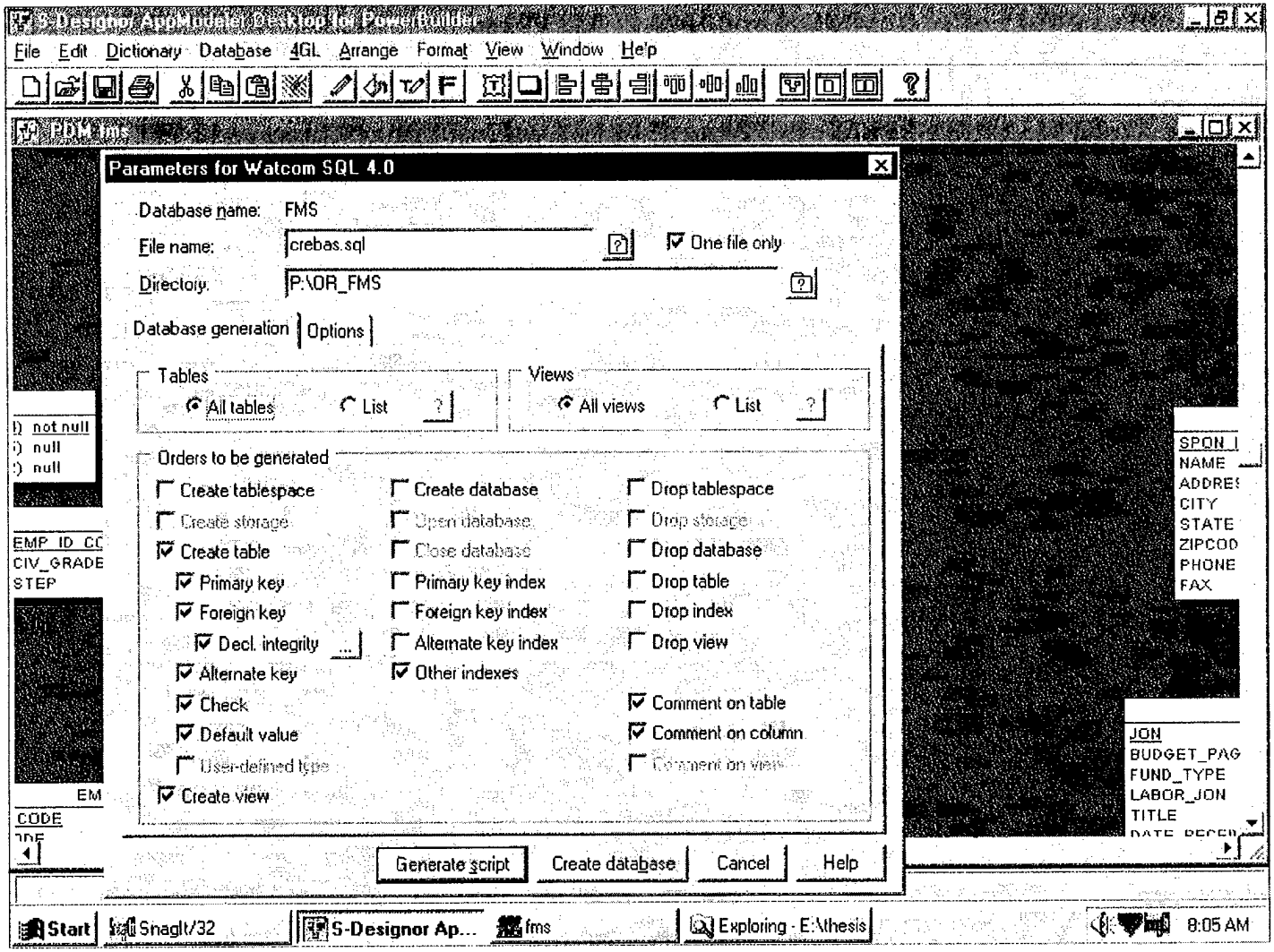


Figure 10. *AppModeler* Database Generation Window

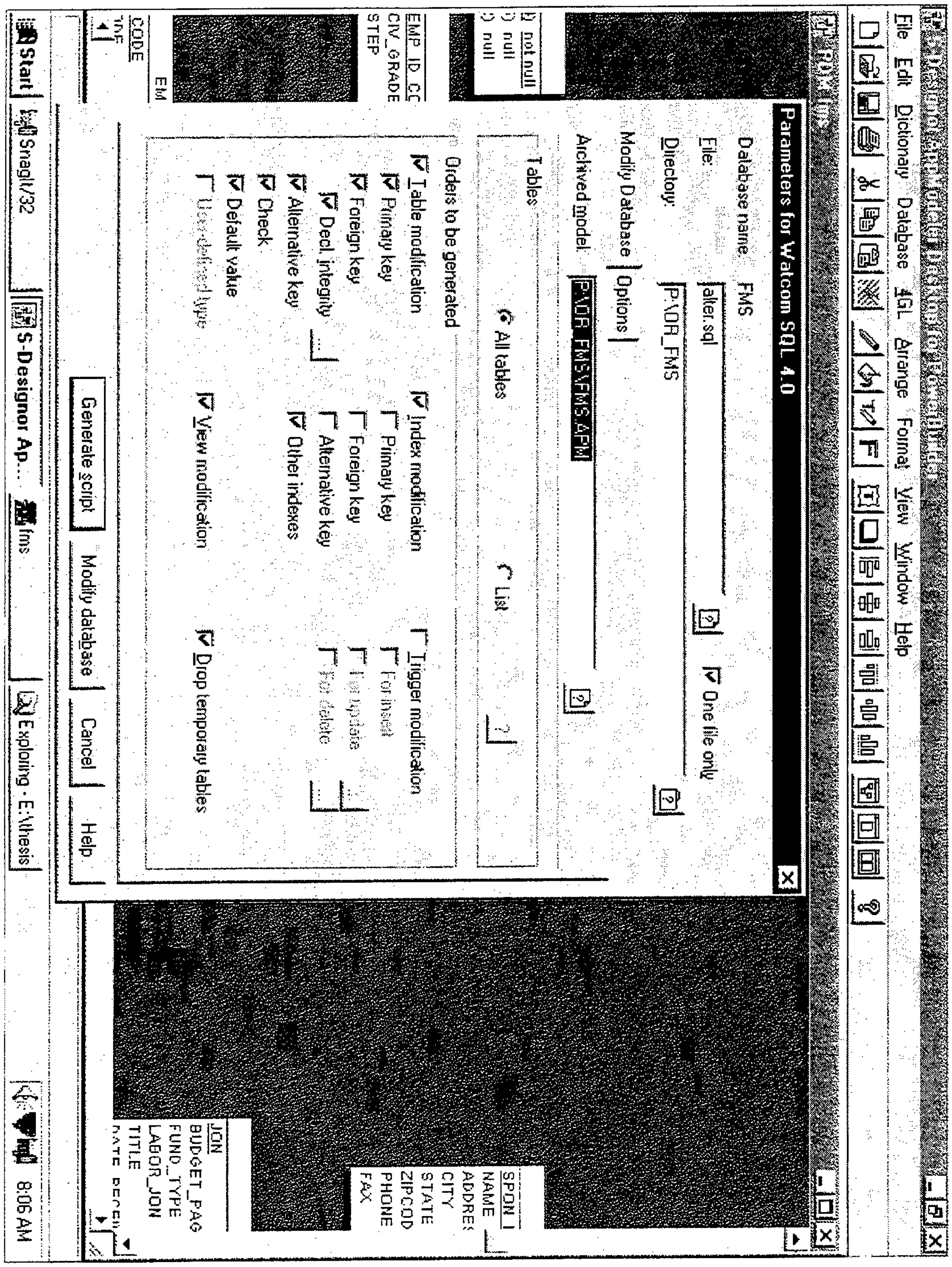


Figure 11. *AppModeler* Database Modification Window

V. CONCLUSIONS AND RECOMMENDATIONS FOR FUTURE STUDY

A. CONCLUSIONS

The prototype Financial Management System currently deployed in the Operations Research Department is nearly a production system which, with some modifications, could be used as an accounting system for all the academic departments at the Naval Postgraduate School to track their financial accounts. The prototype has demonstrated that even though user requirements frequently change, it can be changed to meet new requirements relatively quickly and easily. Comparing the EER diagram in Figure 1 to the physical data model in Figure 2, it is obvious that the design of the FMS changed a great deal over the course of this thesis project. Yet, the majority of changes were implemented within a few a days of the decision to change the design. This quick turn-around for implementing design changes would not have been possible if this project had been prepared using only a programming language such as C++.

The tools used (those contained in *Powersoft Portfolio*) were an invaluable part of this project and very inexpensive when compared to some of the other tools on the market. That is not to say that *Powersoft Portfolio* is the best client/server application development tool set available for those on a tight budget. It did, however, meet the needs of this project and we would recommend it for use by others with similar needs and resources.

Changes and additions need to be made to the FMS. The faculty module has been developed but it needs to be deployed for testing and debugging. Error conditions in the staff module need to be trapped and clear error messages displayed when errors occur. An on-line

help system needs to be added and the users need to be able to easily produce rudimentary reports from the data available. These changes and additions can be made to the system relatively easily using the tools we have available when time permits.

B. RECOMMENDATIONS FOR FUTURE STUDY

The system could be extended to become an automated aid for the academic departments. By extending the database and the application, the system could be used for property management, scheduling classes, and managing other databases used by the departments. This would prevent the same data from being entered multiple times into separate databases. For example, accountable property is tagged with a minor or plant property tag and entered into a database with various attributes about each piece of property. Much of this property is purchased by academic departments from their various accounts and many of the same attributes about this property are stored in the FMS table called OPTAR_REQ as are stored in the property database. Since the FMS is a relational database, it could be made to interface with this property database, i.e., have relationships created with a modified form of the property database tables. Another relation could be created for property that was maintained by staff members at the school, such as computer hardware, so those staff members could keep a record of maintenance performed on the property. Other existing systems at the Naval Postgraduate School such as SACONS (Standard Automated Contracting System) could also be made to interface with the FMS to further reduce multiple entries of the same data and other problems associated with having separate databases that contain essentially the same information. In fact, these existing systems should also be

analyzed for possible changes to maximize the benefits available through the use of client/server database applications.

A "chairman's" module still needs to be developed for the FMS to assist the academic department chairman in planning the expenditure of funds, especially at the beginning of each fiscal year.

A course information database would be another useful addition to the FMS. It could be used to relate planned instruction (courses) to the expenditure of funds for supplies and labor needed to support instruction.

LIST OF REFERENCES

1. Renner, R. B., *Information Requirements Analysis: An Application*, Master's Thesis, Naval Postgraduate School, Monterey, CA, March 1984

2. Booker, R. L., *The AS Financial Reporting System: Some Experience On Prototyping And User Interaction*, Master's Thesis, Naval Postgraduate School, Monterey, CA, March 1986

3. Sexton, T. M., *A Property Management System For The Administrative Sciences Department*, Master's Thesis, Naval Postgraduate School, Monterey, CA, September 1987

4. Ford, N. S., and Zimmon, N. W., *A Data-Based Financial Management Information System (FMIS) For Administrative Sciences Department*, Master's Thesis, Naval Postgraduate School, Monterey, CA, December 1990

5. Ditri, T. A., *Upgrade And Enhancement Of The A. S. Department Financial Management Information System; Development Of The FMIS Property Management Module*, Master's Thesis, Naval Postgraduate School, Monterey, CA, September 1991

6. Elmasri, R., and Navathe, S., *Fundamentals of Database Systems*, The Benjamin/Cummings Publishing Company, Inc., 1989

7. Lewis, C., and Rieman, J., *Task-centered User Interface Design: A Practical Introduction*, not published

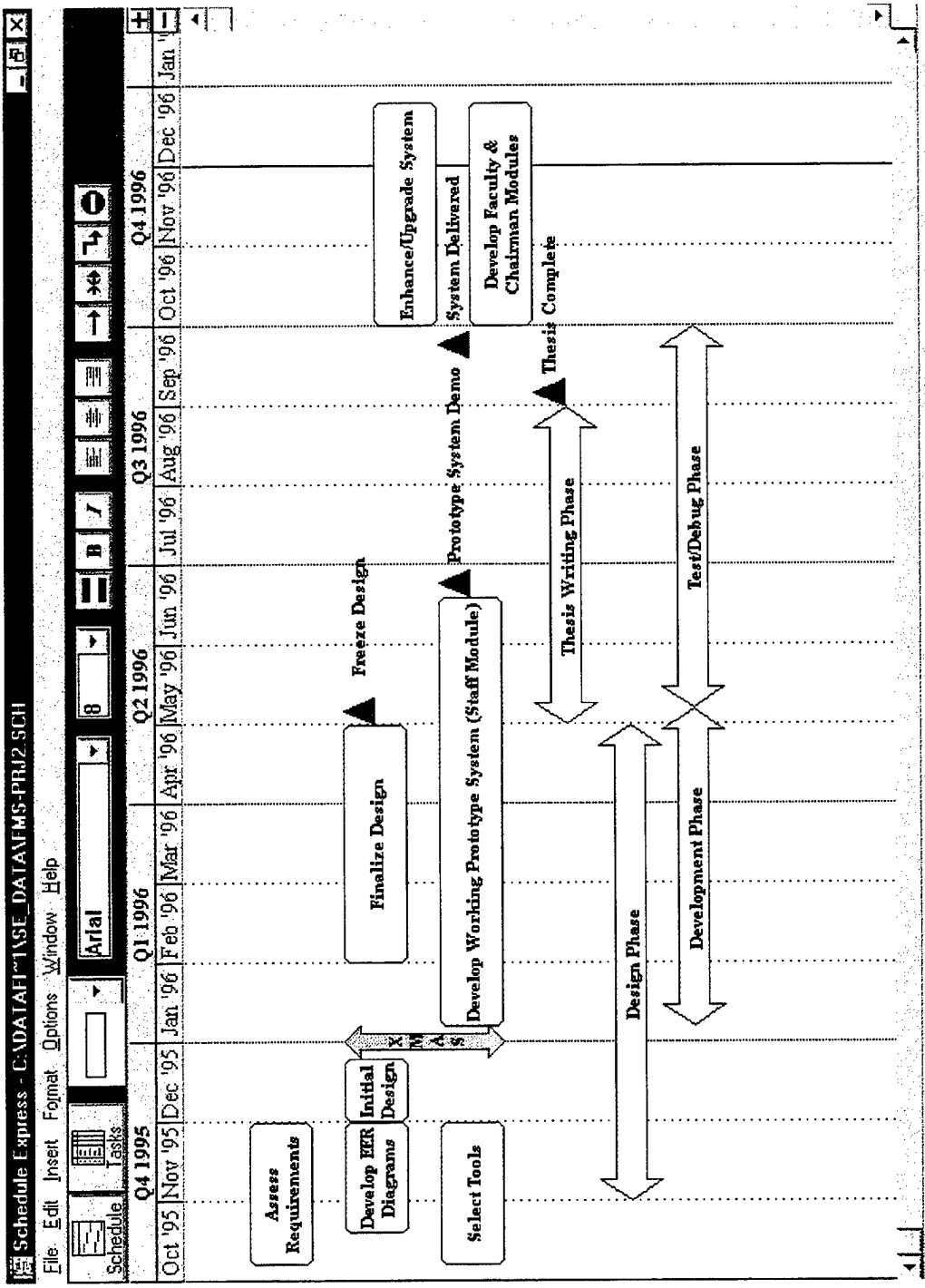
8. Marion, W., *Client/Server Strategies: Implementations In The IBM Environment*, McGraw-Hill, Inc., 1994

9. Shneiderman, B., *Designing the User Interface: Strategies for Effective Human-Computer Interaction*, Addison-Wesley Publishing Company, Inc., 1992

10. Neilsen, J., "The Usability Engineering Life Cycle," *Computer*, pp. 12-22, March 1992

11. Connell, J. L., and Shafer, L. B., *Structured Rapid Prototyping: An Evolutionary Approach to Software Development*, Yourdon Press, 1989

APPENDIX A. PROJECT SCHEDULE



APPENDIX B. FMS DATABASE TRIGGERS

```
%% ============================================================
%%   Database name:  FMS
%%   DBMS name:      Watcom SQL 4.0
%%   Created on:     2/3/97  4:52 PM
%% ============================================================

%  Before insert trigger "tib_account" for table "ACCOUNT"
create trigger tib_account before insert on ACCOUNT
referencing new as new_ins for each row
begin
    declare user_defined_exception exception for SQLSTATE '99999';
    declare found integer;

    %  Parent "SPONSOR" must exist when inserting a child in "ACCOUNT"
    if (new_ins.SPON_ID_CODE is not null) then
    begin
       set found = 0;
       select 1
        into  found
        from  dummy
       where  exists (select 1
                      from  SPONSOR
                     where  SPON_ID_CODE = new_ins.SPON_ID_CODE);
       if found <> 1 then
          signal user_defined_exception
       end if;
    end
    end if;
end
/

%  After insert trigger "tia_account" for table "ACCOUNT"
create trigger tia_account after insert on ACCOUNT
referencing new as new_ins for each row
begin
  call CALC_BAL_CONTRACT(new_ins.JON,'M');
  call CALC_BAL_CONTRACT(new_ins.JON,'I');
  call CALC_BAL_CONTRACT(new_ins.JON,'O');
  call CALC_BAL_FAC_LABOR(new_ins.JON);
  call CALC_BAL_SPT_LABOR(new_ins.JON);
  call CALC_BAL_OPTAR(new_ins.JON);
  call CALC_BAL_TRAV(new_ins.JON);
end
/

%  Update trigger "tua_account" for table "ACCOUNT"
create trigger tua_account after update of INIT_FAC_LABOR_$,
                                           INIT_SPT_LABOR_$,
                                           INIT_TRAVEL_$,
```

```
                                        INIT_OPTAR_$,
                                        INIT_CONT_MIPR,
                                        INIT_CONT_IPA,
                                        INIT_CONT_OTH
on ACCOUNT
referencing new as new_upd old as old_upd for each row
begin
  declare user_defined_exception exception for SQLSTATE '99999';
  declare found integer;
  call CALC_BAL_CONTRACT(new_upd.JON,'M');
  call CALC_BAL_CONTRACT(new_upd.JON,'I');
  call CALC_BAL_CONTRACT(new_upd.JON,'O');
  call CALC_BAL_FAC_LABOR(new_upd.JON);
  call CALC_BAL_SPT_LABOR(new_upd.JON);
  call CALC_BAL_OPTAR(new_upd.JON);
  call CALC_BAL_TRAV(new_upd.JON);
end
/

%  Before insert trigger "tib_adp_proj_info" for table "ADP_PROJ_INFO"
create trigger tib_adp_proj_info before insert on ADP_PROJ_INFO
referencing new as new_ins for each row
begin
    declare user_defined_exception exception for SQLSTATE '99999';
    declare found integer;

      %   Parent "DEPARTMENT" must exist when inserting a child in
"ADP_PROJ_INFO"
    if (new_ins.DEPT_CODE is not null) then
    begin
       set found = 0;
       select 1
        into  found
        from  dummy
       where  exists (select 1
                        from  DEPARTMENT
                       where  DEPT_CODE = new_ins.DEPT_CODE);
       if found <> 1 then
          signal user_defined_exception
       end if;
    end
    end if;

      %   Parent "EMPLOYEE" must exist when inserting a child in
"ADP_PROJ_INFO"
    if (new_ins.PROJ_MGR_CODE is not null) then
    begin
       set found = 0;
       select 1
        into  found
        from  dummy
       where  exists (select 1
```

```
                    from  EMPLOYEE
                   where  EMP_ID_CODE = new_ins.PROJ_MGR_CODE);
      if found <> 1 then
         signal user_defined_exception
      end if;
   end
   end if;

      %   Parent "EMPLOYEE" must exist when inserting a child in
"ADP_PROJ_INFO"
   if (new_ins.POC_CODE is not null) then
   begin
      set found = 0;
      select 1
       into  found
       from  dummy
      where  exists (select 1
                    from  EMPLOYEE
                   where  EMP_ID_CODE = new_ins.POC_CODE);
      if found <> 1 then
         signal user_defined_exception
      end if;
   end
   end if;
end
/

%  Before insert trigger "tib_contracts" for table "CONTRACTS"
create trigger tib_contracts before insert on CONTRACTS
referencing new as new_ins for each row
begin
   declare user_defined_exception exception for SQLSTATE '99999';
   declare found integer;

   %  Parent "ACCOUNT" must exist when inserting a child in "CONTRACTS"
   if (new_ins.JON is not null) then
   begin
      set found = 0;
      select 1
       into  found
       from  dummy
      where  exists (select 1
                    from  ACCOUNT
                   where  JON = new_ins.JON);
      if found <> 1 then
         signal user_defined_exception
      end if;
   end
   end if;

   %  Parent "EMPLOYEE" must exist when inserting a child in "CONTRACTS"
   if (new_ins.REQUESTER is not null) then
```

```
    begin
       set found = 0;
       select 1
        into  found
        from  dummy
       where  exists (select 1
                        from  EMPLOYEE
                       where  EMP_ID_CODE = new_ins.REQUESTER);
       if found <> 1 then
          signal user_defined_exception
       end if;
    end
    end if;
end
/

%  After insert trigger "tia_contracts" for table "CONTRACTS"
create trigger tia_contracts after insert on CONTRACTS
referencing new as new_ins for each row
begin
  declare user_defined_exception exception for SQLSTATE '99999';

  call CALC_BAL_CONTRACT(new_ins.JON,new_ins.CONTRACT_TYPE)

end
/

%  Before update trigger "tub_contracts" for table "CONTRACTS"
create trigger tub_contracts before update of JON,
                                              CONTRACT_TYPE,
                                              REQUESTER,
                                              DOC_#
on CONTRACTS
referencing new as new_upd old as old_upd for each row
begin
    declare user_defined_exception exception for SQLSTATE '99999';
    declare found integer;

    %  Parent "ACCOUNT" must exist when updating a child in "CONTRACTS"
    if (new_upd.JON is not null and
        ((old_upd.JON is null) or
         (new_upd.JON <> old_upd.JON))) then
    begin
       set found = 0;
       select 1
        into  found
        from  dummy
       where  exists (select 1
                        from  ACCOUNT
                       where  JON = new_upd.JON);
       if found <> 1 then
          signal user_defined_exception
```

```
      end if;
   end
   end if;

   %  Parent "EMPLOYEE" must exist when updating a child in "CONTRACTS"
   if (new_upd.REQUESTER is not null and
       ((old_upd.REQUESTER is null) or
        (new_upd.REQUESTER <> old_upd.REQUESTER))) then
   begin
      set found = 0;
      select 1
       into  found
       from  dummy
      where  exists (select 1
                     from  EMPLOYEE
                    where  EMP_ID_CODE = new_upd.REQUESTER);
      if found <> 1 then
         signal user_defined_exception
      end if;
   end
   end if;

   %  Cannot modify parent code of "EMPLOYEE" in child "CONTRACTS"
   if ((new_upd.REQUESTER is null and old_upd.REQUESTER is not null) or
       new_upd.REQUESTER <> old_upd.REQUESTER ) then
      signal user_defined_exception
   end if;
end
/

%  Update trigger "tua_contracts" for table "CONTRACTS"
create trigger tua_contracts after update of JON,
                                             CONTRACT_TYPE,
                                             CONTRACTOR_ID,
                                             PROJ_COST,
                                             ACTUAL_COST
on CONTRACTS
referencing new as new_upd old as old_upd for each row
begin
  declare user_defined_exception exception for SQLSTATE '99999';
  declare found integer;


  call CALC_BAL_CONTRACT(new_upd.JON,new_upd.CONTRACT_TYPE)

end
/

%  After delete trigger "tda_contracts" for table "CONTRACTS"
create trigger tda_contracts after delete on CONTRACTS
referencing old as old_del for each row
begin
```

```
  declare user_defined_exception exception for SQLSTATE '99999';
  declare found integer;


  call CALC_BAL_CONTRACT(old_del.JON,old_del.CONTRACT_TYPE)

end
/

%  Before insert trigger "tib_employee" for table "EMPLOYEE"
create trigger tib_employee before insert on EMPLOYEE
referencing new as new_ins for each row
begin
    declare user_defined_exception exception for SQLSTATE '99999';
    declare found integer;

    %  Parent "DEPARTMENT" must exist when inserting a child in "EMPLOYEE"
    if (new_ins.DEPT_CODE is not null) then
    begin
       set found = 0;
       select 1
        into  found
        from  dummy
       where  exists (select 1
                        from  DEPARTMENT
                       where  DEPT_CODE = new_ins.DEPT_CODE);
       if found <> 1 then
          signal user_defined_exception
       end if;
    end
    end if;
end
/

%  Before insert trigger "tib_faculty" for table "FACULTY"
create trigger tib_faculty before insert on FACULTY
referencing new as new_ins for each row
begin
    declare user_defined_exception exception for SQLSTATE '99999';
    declare found integer;

    %  Parent "EMPLOYEE" must exist when inserting a child in "FACULTY"
    if (new_ins.EMP_ID_CODE is not null) then
    begin
       set found = 0;
       select 1
        into  found
        from  dummy
       where  exists (select 1
                        from  EMPLOYEE
                       where  EMP_ID_CODE = new_ins.EMP_ID_CODE);
       if found <> 1 then
```

```
            signal user_defined_exception
        end if;
    end
    end if;
end
/

%  Before insert trigger "tib_labor_chgs" for table "LABOR_CHGS"
create trigger tib_labor_chgs before insert on LABOR_CHGS
referencing new as new_ins for each row
begin
    declare user_defined_exception exception for SQLSTATE '99999';
    declare found integer;

       %    Parent  "LABOR_LES"  must  exist  when  inserting  a  child  in
"LABOR_CHGS"
    if (new_ins.EMP_ID_CODE is not null and
        new_ins.PPE_DATE is not null) then
    begin
       set found = 0;
       select 1
        into  found
        from  dummy
       where  exists (select 1
                      from  LABOR_LES
                     where  EMP_ID_CODE = new_ins.EMP_ID_CODE
                      and   PPE_DATE = new_ins.PPE_DATE);
       if found <> 1 then
          signal user_defined_exception
       end if;
    end
    end if;

    %  Parent "ACCOUNT" must exist when inserting a child in "LABOR_CHGS"
    if (new_ins.JON is not null) then
    begin
       set found = 0;
       select 1
        into  found
        from  dummy
       where  exists (select 1
                      from  ACCOUNT
                     where  JON = new_ins.JON);
       if found <> 1 then
          signal user_defined_exception
       end if;
    end
    end if;

    %  Parent "EMPLOYEE" must exist when inserting a child in "LABOR_CHGS"
    if (new_ins.EMP_ID_CODE is not null) then
    begin
```

```
        set found = 0;
        select 1
         into  found
         from  dummy
        where  exists (select 1
                       from  EMPLOYEE
                      where  EMP_ID_CODE = new_ins.EMP_ID_CODE);
        if found <> 1 then
           signal user_defined_exception
        end if;
     end
     end if;
end
/

%  After insert trigger "tia_labor_chgs" for table "LABOR_CHGS"
create trigger tia_labor_chgs after insert on LABOR_CHGS
referencing new as new_ins for each row
begin
  declare user_defined_exception exception for SQLSTATE '99999';
  declare found integer;
  declare emp_cat char(1);
  declare jon_type char(2);
  declare base_sal numeric(10,2);
  declare hourly_rate numeric (7,2);
  declare hourly_ot_rate numeric(7,2);
  declare otm_cap numeric(7,2);
  declare yr_hrs integer;
  declare rr_ot_fac numeric(6,4);
  declare sal_eff date;
  declare acc_rate decimal(3,2);

  select OT_CAP into otm_cap from FMS_CFG;
  select YR_LABOR_HRS into yr_hrs from FMS_CFG;
  select RR_OT_RATE_FACT into rr_ot_fac from FMS_CFG;

  %Calculate the "TOTALCHG" field
  if (new_ins.EMP_ID_CODE is not null) then
  begin
    set found=0;
    select 1
      into found
      from dummy
    where exists (select 1
      from EMPLOYEE
      where EMP_ID_CODE=new_ins.EMP_ID_CODE);

    select EFF_SAL_DATE into sal_eff from EMPLOYEE
      where new_ins.EMP_ID_CODE=EMPLOYEE.EMP_ID_CODE;

    if (new_ins.PPE_DATE >= sal_eff) then
    begin
```

```
  select BASE_SALARY into base_sal from EMPLOYEE
    where new_ins.EMP_ID_CODE=EMPLOYEE.EMP_ID_CODE;
  select ACCEL_RATE into acc_rate from EMPLOYEE
    where new_ins.EMP_ID_CODE=EMPLOYEE.EMP_ID_CODE;
end
else
begin
  select BASE_SALARY into base_sal from SALARY_HISTORY
    where new_ins.EMP_ID_CODE=SALARY_HISTORY.EMP_ID_CODE
     and new_ins.PPE_DATE >= SALARY_HISTORY.BEGIN_DATE
     and new_ins.PPE_DATE <= SALARY_HISTORY.END_DATE;
  select ACCEL_RATE into acc_rate from SALARY_HISTORY
    where new_ins.EMP_ID_CODE=SALARY_HISTORY.EMP_ID_CODE
     and new_ins.PPE_DATE >= SALARY_HISTORY.BEGIN_DATE
     and new_ins.PPE_DATE <= SALARY_HISTORY.END_DATE;
end
end if;

set hourly_rate=base_sal/yr_hrs;

if ((hourly_rate*1.5) > otm_cap) then
  set hourly_ot_rate=otm_cap
else
  set hourly_ot_rate=hourly_rate*1.5
end if;

select FUND_TYPE into jon_type from ACCOUNT
  where new_ins.JON=ACCOUNT.JON;

select CATEGORY into emp_cat from EMPLOYEE
  where new_ins.EMP_ID_CODE=EMPLOYEE.EMP_ID_CODE;

if (jon_type='RR') then
begin
  if (emp_cat='F') then
    update LABOR_CHGS,EMPLOYEE
      set TOTAL_CHG=(HOURS*hourly_rate*acc_rate)
    where LABOR_CHGS.EMP_ID_CODE=new_ins.EMP_ID_CODE
      and LABOR_CHGS.PPE_DATE=new_ins.PPE_DATE
      and LABOR_CHGS.JON=new_ins.JON
      and new_ins.EMP_ID_CODE=EMPLOYEE.EMP_ID_CODE
  else
    if (emp_cat='S') then
      update LABOR_CHGS,EMPLOYEE
        set TOTAL_CHG=(HOURS*hourly_rate*acc_rate)+
                      (OT_HOURS*hourly_ot_rate*rr_ot_fac)
    where LABOR_CHGS.EMP_ID_CODE=new_ins.EMP_ID_CODE
      and LABOR_CHGS.PPE_DATE=new_ins.PPE_DATE
      and LABOR_CHGS.JON=new_ins.JON
      and new_ins.EMP_ID_CODE=EMPLOYEE.EMP_ID_CODE
    end if
  end if
```

```
    end
    else
    begin
      if (emp_cat='F') then
        update LABOR_CHGS
          set TOTAL_CHG=(HOURS*hourly_rate)
        where LABOR_CHGS.EMP_ID_CODE=new_ins.EMP_ID_CODE
          and LABOR_CHGS.PPE_DATE=new_ins.PPE_DATE
          and LABOR_CHGS.JON=new_ins.JON
      else
        if (emp_cat='S') then
          update LABOR_CHGS
            set TOTAL_CHG=(HOURS*hourly_rate)+
                          (OT_HOURS*hourly_ot_rate)
        where LABOR_CHGS.EMP_ID_CODE=new_ins.EMP_ID_CODE
          and LABOR_CHGS.PPE_DATE=new_ins.PPE_DATE
          and LABOR_CHGS.JON=new_ins.JON
        end if
      end if
    end
    end if;

    if (emp_cat='F') then
      call CALC_BAL_FAC_LABOR(new_ins.JON)
    else
      if (emp_cat='S') then
        call CALC_BAL_SPT_LABOR(new_ins.JON)
      end if
    end if;

    if (found <> 1) then
      signal user_defined_exception
    end if;
  end
  end if;
end
/

%  Update trigger "tua_labor_chgs" for table "LABOR_CHGS"
create trigger tua_labor_chgs after update of EMP_ID_CODE,
                                              PPE_DATE,
                                              JON,
                                              HOURS,
                                              OT_HOURS
on LABOR_CHGS
referencing new as new_upd old as old_upd for each row
begin
  declare user_defined_exception exception for SQLSTATE '99999';
  declare found integer;
  declare emp_cat char(1);
  declare jon_type char(2);
  declare base_sal numeric(10,2);
```

```
declare hourly_rate numeric (7,2);
declare hourly_ot_rate numeric(7,2);
declare otm_cap numeric(7,2);
declare yr_hrs integer;
declare rr_ot_fac numeric(6,4);
declare sal_eff date;
declare acc_rate decimal(3,2);


select OT_CAP into otm_cap from FMS_CFG;
select YR_LABOR_HRS into yr_hrs from FMS_CFG;
select RR_OT_RATE_FACT into rr_ot_fac from FMS_CFG;

%Calculate the "TOTALCHG" field
if ((new_upd.HOURS<>old_upd.HOURS) or
   (new_upd.OT_HOURS<>old_upd.OT_HOURS)) then
begin
  set found=0;
  select 1
    into found
    from dummy
  where exists (select 1
    from EMPLOYEE
    where EMP_ID_CODE=new_upd.EMP_ID_CODE);

  select EFF_SAL_DATE into sal_eff from EMPLOYEE
    where new_upd.EMP_ID_CODE=EMPLOYEE.EMP_ID_CODE;

  if (new_upd.PPE_DATE >= sal_eff) then
  begin
    select BASE_SALARY into base_sal from EMPLOYEE
      where new_upd.EMP_ID_CODE=EMPLOYEE.EMP_ID_CODE;
    select ACCEL_RATE into acc_rate from EMPLOYEE
      where new_upd.EMP_ID_CODE=EMPLOYEE.EMP_ID_CODE;
  end
  else
  begin
    select BASE_SALARY into base_sal from SALARY_HISTORY
      where new_upd.EMP_ID_CODE=SALARY_HISTORY.EMP_ID_CODE
       and new_upd.PPE_DATE >= SALARY_HISTORY.BEGIN_DATE
       and new_upd.PPE_DATE <= SALARY_HISTORY.END_DATE;
    select ACCEL_RATE into acc_rate from SALARY_HISTORY
      where new_upd.EMP_ID_CODE=SALARY_HISTORY.EMP_ID_CODE
       and new_upd.PPE_DATE >= SALARY_HISTORY.BEGIN_DATE
       and new_upd.PPE_DATE <= SALARY_HISTORY.END_DATE;
  end
  end if;

  set hourly_rate=base_sal/yr_hrs;

  if ((hourly_rate*1.5) > otm_cap) then
    set hourly_ot_rate=otm_cap
```

```
else
  set hourly_ot_rate=hourly_rate*1.5
end if;

select FUND_TYPE into jon_type from ACCOUNT
  where new_upd.JON=ACCOUNT.JON;

select CATEGORY into emp_cat from EMPLOYEE
  where new_upd.EMP_ID_CODE=EMPLOYEE.EMP_ID_CODE;

if (jon_type='RR') then
begin
  if (emp_cat='F') then
    update LABOR_CHGS,EMPLOYEE
      set TOTAL_CHG=(HOURS*hourly_rate*acc_rate)
    where LABOR_CHGS.EMP_ID_CODE=new_upd.EMP_ID_CODE
      and LABOR_CHGS.PPE_DATE=new_upd.PPE_DATE
      and LABOR_CHGS.JON=new_upd.JON
      and new_upd.EMP_ID_CODE=EMPLOYEE.EMP_ID_CODE
  else
    if (emp_cat='S') then
      update LABOR_CHGS,EMPLOYEE
        set TOTAL_CHG=(HOURS*hourly_rate*acc_rate)+
                      (OT_HOURS*hourly_ot_rate*rr_ot_fac)
      where LABOR_CHGS.EMP_ID_CODE=new_upd.EMP_ID_CODE
        and LABOR_CHGS.PPE_DATE=new_upd.PPE_DATE
        and LABOR_CHGS.JON=new_upd.JON
        and new_upd.EMP_ID_CODE=EMPLOYEE.EMP_ID_CODE
    end if
  end if
end
else
begin
  if (emp_cat='F') then
    update LABOR_CHGS
      set TOTAL_CHG=(HOURS*hourly_rate)
    where LABOR_CHGS.EMP_ID_CODE=new_upd.EMP_ID_CODE
      and LABOR_CHGS.PPE_DATE=new_upd.PPE_DATE
      and LABOR_CHGS.JON=new_upd.JON
  else
    if (emp_cat='S') then
      update LABOR_CHGS
        set TOTAL_CHG=(HOURS*hourly_rate)+
                      (OT_HOURS*hourly_ot_rate)
      where LABOR_CHGS.EMP_ID_CODE=new_upd.EMP_ID_CODE
        and LABOR_CHGS.PPE_DATE=new_upd.PPE_DATE
        and LABOR_CHGS.JON=new_upd.JON
    end if
  end if
end
end if;
```

```
    if (emp_cat='F') then
      call CALC_BAL_FAC_LABOR(new_upd.JON)
    else
      if (emp_cat='S') then
        call CALC_BAL_SPT_LABOR(new_upd.JON)
      end if
    end if;

    if (found <> 1) then
      signal user_defined_exception
    end if;
  end
  end if;
end
/

%  After delete trigger "tda_labor_chgs" for table "LABOR_CHGS"
create trigger tda_labor_chgs after delete on LABOR_CHGS
referencing old as old_del for each row
begin
  declare user_defined_exception exception for SQLSTATE '99999';
  declare found integer;
  declare emp_cat char(1);


  select CATEGORY into emp_cat from EMPLOYEE
    where old_del.EMP_ID_CODE=EMPLOYEE.EMP_ID_CODE;

  if (emp_cat='F') then
    call CALC_BAL_FAC_LABOR(old_del.JON)
  elseif (emp_cat='S') then
    call CALC_BAL_SPT_LABOR(old_del.JON)
  end if;

end
/

%  Before insert trigger "tib_labor_les" for table "LABOR_LES"
create trigger tib_labor_les before insert on LABOR_LES
referencing new as new_ins for each row
begin
    declare user_defined_exception exception for SQLSTATE '99999';
    declare found integer;

    %  Parent "EMPLOYEE" must exist when inserting a child in "LABOR_LES"
    if (new_ins.EMP_ID_CODE is not null) then
    begin
       set found = 0;
       select 1
        into  found
        from  dummy
       where  exists (select 1
```

```
                        from  EMPLOYEE
                       where  EMP_ID_CODE = new_ins.EMP_ID_CODE);
        if found <> 1 then
           signal user_defined_exception
        end if;
     end
     end if;
end
/

%  Before insert trigger "tib_military" for table "MILITARY"
create trigger tib_military before insert on MILITARY
referencing new as new_ins for each row
begin
     declare user_defined_exception exception for SQLSTATE '99999';
     declare found integer;

     %  Parent "EMPLOYEE" must exist when inserting a child in "MILITARY"
     if (new_ins.EMP_ID_CODE is not null) then
     begin
        set found = 0;
        select 1
         into  found
         from  dummy
        where  exists (select 1
                        from  EMPLOYEE
                       where  EMP_ID_CODE = new_ins.EMP_ID_CODE);
        if found <> 1 then
           signal user_defined_exception
        end if;
     end
     end if;
end
/

%  Before insert trigger "tib_optar_req" for table "OPTAR_REQ"
create trigger tib_optar_req before insert on OPTAR_REQ
referencing new as new_ins for each row
begin
     declare user_defined_exception exception for SQLSTATE '99999';
     declare found integer;

     %  Parent "EMPLOYEE" must exist when inserting a child in "OPTAR_REQ"
     if (new_ins.EMP_ID_CODE is not null) then
     begin
        set found = 0;
        select 1
         into  found
         from  dummy
        where  exists (select 1
                        from  EMPLOYEE
                       where  EMP_ID_CODE = new_ins.EMP_ID_CODE);
```

```
      if found <> 1 then
         signal user_defined_exception
      end if;
   end
   end if;

   %  Parent "ACCOUNT" must exist when inserting a child in "OPTAR_REQ"
   if (new_ins.JON is not null) then
   begin
      set found = 0;
      select 1
       into  found
       from  dummy
      where  exists (select 1
                      from  ACCOUNT
                     where  JON = new_ins.JON);
      if found <> 1 then
         signal user_defined_exception
      end if;
   end
   end if;

     %   Parent "ADP_PROJ_INFO" must exist when inserting a child in
"OPTAR_REQ"
   if (new_ins.ADP_PROJ_# is not null) then
   begin
      set found = 0;
      select 1
       into  found
       from  dummy
      where  exists (select 1
                      from  ADP_PROJ_INFO
                     where  ADP_PROJ_# = new_ins.ADP_PROJ_#);
      if found <> 1 then
         signal user_defined_exception
      end if;
   end
   end if;
end
/

%  After insert trigger "tia_optar_req" for table "OPTAR_REQ"
create trigger tia_optar_req after insert on OPTAR_REQ
referencing new as new_ins for each row
begin
  declare user_defined_exception exception for SQLSTATE '99999';

  call CALC_BAL_OPTAR(new_ins.JON);

end
/
```

```
%  Update trigger "tua_optar_req" for table "OPTAR_REQ"
create trigger tua_optar_req after update of JON,
                                             EMP_ID_CODE,
                                             DOC_#,
                                             PROJ_COST,
                                             ACTUAL_COST,
                                             ADP_PROJ_#
on OPTAR_REQ
referencing new as new_upd old as old_upd for each row
begin
  declare user_defined_exception exception for SQLSTATE '99999';
  declare found integer;


  call CALC_BAL_OPTAR(new_upd.JON);

end
/

%  After delete trigger "tda_optar_req" for table "OPTAR_REQ"
create trigger tda_optar_req after delete on OPTAR_REQ
referencing old as old_del for each row
begin
  declare user_defined_exception exception for SQLSTATE '99999';
  declare found integer;


  call CALC_BAL_OPTAR(old_del.JON);

end
/

%  Before insert trigger "tib_other_leave" for table "OTHER_LEAVE"
create trigger tib_other_leave before insert on OTHER_LEAVE
referencing new as new_ins for each row
begin
    declare user_defined_exception exception for SQLSTATE '99999';
    declare found integer;

       %    Parent  "LABOR_LES"  must  exist  when  inserting  a  child  in
"OTHER_LEAVE"
    if (new_ins.EMP_ID_CODE is not null and
        new_ins.PPE_DATE is not null) then
    begin
       set found = 0;
       select 1
        into  found
        from  dummy
       where  exists (select 1
                        from  LABOR_LES
                       where  EMP_ID_CODE = new_ins.EMP_ID_CODE
                        and   PPE_DATE = new_ins.PPE_DATE);
```

```
        if found <> 1 then
           signal user_defined_exception
        end if;
     end
     end if;

       %   Parent "OTHER_LV_TYPE" must exist when inserting a child in
"OTHER_LEAVE"
     if (new_ins.TYPE is not null) then
     begin
        set found = 0;
        select 1
         into  found
         from  dummy
        where  exists (select 1
                       from  OTHER_LV_TYPE
                      where  OTHER_LV_TYPE_CODE = new_ins.TYPE);
        if found <> 1 then
           signal user_defined_exception
        end if;
     end
     end if;
end
/

%  Before update trigger "tub_other_leave" for table "OTHER_LEAVE"
create trigger tub_other_leave before update of EMP_ID_CODE,
                                                PPE_DATE,
                                                TYPE
on OTHER_LEAVE
referencing new as new_upd old as old_upd for each row
begin
     declare user_defined_exception exception for SQLSTATE '99999';
     declare found integer;

        %    Parent  "LABOR_LES"  must  exist  when  updating  a  child  in
"OTHER_LEAVE"
     if (new_upd.EMP_ID_CODE is not null and
         new_upd.PPE_DATE is not null and
         ((old_upd.EMP_ID_CODE is null and
           old_upd.PPE_DATE is null) or
          (new_upd.EMP_ID_CODE <> old_upd.EMP_ID_CODE or
           new_upd.PPE_DATE <> old_upd.PPE_DATE))) then
     begin
        set found = 0;
        select 1
         into  found
         from  dummy
        where  exists (select 1
                       from  LABOR_LES
                      where  EMP_ID_CODE = new_upd.EMP_ID_CODE
                       and   PPE_DATE = new_upd.PPE_DATE);
```

```
        if found <> 1 then
           signal user_defined_exception
        end if;
     end
     end if;

       %   Parent "OTHER_LV_TYPE" must exist when updating a child in
"OTHER_LEAVE"
     if (new_upd.TYPE is not null and
         ((old_upd.TYPE is null) or
          (new_upd.TYPE <> old_upd.TYPE))) then
     begin
        set found = 0;
        select 1
         into  found
         from  dummy
        where  exists (select 1
                       from  OTHER_LV_TYPE
                      where  OTHER_LV_TYPE_CODE = new_upd.TYPE);
        if found <> 1 then
           signal user_defined_exception
        end if;
     end
     end if;

     %  Cannot modify parent code of "OTHER_LV_TYPE" in child "OTHER_LEAVE"
     if ((new_upd.TYPE is null and old_upd.TYPE is not null) or
         new_upd.TYPE <> old_upd.TYPE ) then
        signal user_defined_exception
     end if;
end
/

%  Before insert trigger "tib_pi" for table "PI"
create trigger tib_pi before insert on PI
referencing new as new_ins for each row
begin
     declare user_defined_exception exception for SQLSTATE '99999';
     declare found integer;

     %  Parent "EMPLOYEE" must exist when inserting a child in "PI"
     if (new_ins.EMP_ID_CODE is not null) then
     begin
        set found = 0;
        select 1
         into  found
         from  dummy
        where  exists (select 1
                       from  EMPLOYEE
                      where  EMP_ID_CODE = new_ins.EMP_ID_CODE);
        if found <> 1 then
           signal user_defined_exception
```

```
        end if;
    end
    end if;

    %  Parent "ACCOUNT" must exist when inserting a child in "PI"
    if (new_ins.JON is not null) then
    begin
        set found = 0;
        select 1
         into  found
         from  dummy
        where  exists (select 1
                       from  ACCOUNT
                      where  JON = new_ins.JON);
        if found <> 1 then
           signal user_defined_exception
        end if;
    end
    end if;
end
/

%  Before insert trigger "tib_salary_history" for table "SALARY_HISTORY"
create trigger tib_salary_history before insert on SALARY_HISTORY
referencing new as new_ins for each row
begin
    declare user_defined_exception exception for SQLSTATE '99999';
    declare found integer;

        %    Parent  "EMPLOYEE"  must  exist  when  inserting  a  child  in
"SALARY_HISTORY"
    if (new_ins.EMP_ID_CODE is not null) then
    begin
        set found = 0;
        select 1
         into  found
         from  dummy
        where  exists (select 1
                       from  EMPLOYEE
                      where  EMP_ID_CODE = new_ins.EMP_ID_CODE);
        if found <> 1 then
           signal user_defined_exception
        end if;
    end
    end if;
end
/

%  Before insert trigger "tib_staff" for table "STAFF"
create trigger tib_staff before insert on STAFF
referencing new as new_ins for each row
begin
```

```
    declare user_defined_exception exception for SQLSTATE '99999';
    declare found integer;

    %  Parent "EMPLOYEE" must exist when inserting a child in "STAFF"
    if (new_ins.EMP_ID_CODE is not null) then
    begin
       set found = 0;
       select 1
        into  found
        from  dummy
       where  exists (select 1
                        from  EMPLOYEE
                       where  EMP_ID_CODE = new_ins.EMP_ID_CODE);
       if found <> 1 then
          signal user_defined_exception
       end if;
    end
    end if;
end
/

%  Before insert trigger "tib_travel" for table "TRAVEL"
create trigger tib_travel before insert on TRAVEL
referencing new as new_ins for each row
begin
    declare user_defined_exception exception for SQLSTATE '99999';
    declare found integer;

    %  Parent "ACCOUNT" must exist when inserting a child in "TRAVEL"
    if (new_ins.JON is not null) then
    begin
       set found = 0;
       select 1
        into  found
        from  dummy
       where  exists (select 1
                        from  ACCOUNT
                       where  JON = new_ins.JON);
       if found <> 1 then
          signal user_defined_exception
       end if;
    end
    end if;
end
/

%  After insert trigger "tia_travel" for table "TRAVEL"
create trigger tia_travel after insert on TRAVEL
referencing new as new_ins for each row
begin
  declare user_defined_exception exception for SQLSTATE '99999';
```

```
  call CALC_BAL_TRAV(new_ins.JON);

end
/

%  Update trigger "tua_travel" for table "TRAVEL"
create trigger tua_travel after update of TO#,
                                          PROJ_COST,
                                          ACTUAL_COST,
                                          JON
on TRAVEL
referencing new as new_upd old as old_upd for each row
begin
  declare user_defined_exception exception for SQLSTATE '99999';
  declare found integer;


  call CALC_BAL_TRAV(new_upd.JON);

end
/

%  After delete trigger "tda_travel" for table "TRAVEL"
create trigger tda_travel after delete on TRAVEL
referencing old as old_del for each row
begin
  declare user_defined_exception exception for SQLSTATE '99999';
  declare found integer;


  call CALC_BAL_TRAV(old_del.JON);

end
/

%  Before insert trigger "tib_travel_requests" for table "TRAVEL_REQUESTS"
create trigger tib_travel_requests before insert on TRAVEL_REQUESTS
referencing new as new_ins for each row
begin
    declare user_defined_exception exception for SQLSTATE '99999';
    declare found integer;

        %    Parent  "TRAVEL"  must  exist  when  inserting  a  child  in
"TRAVEL_REQUESTS"
    if (new_ins.TO# is not null) then
    begin
       set found = 0;
       select 1
        into  found
        from  dummy
       where  exists (select 1
                      from  TRAVEL
```

```
                    where  TO# = new_ins.TO#);
        if found <> 1 then
           signal user_defined_exception
        end if;
     end
     end if;
end
/
```

APPENDIX C. FMS DATABASE STORED PROCEDURES

```
%*********************************************************%
% Procedure CALC_BAL_CONTRACT
%*********************************************************%
create procedure %PROC% (IN jo_num char(5), cont_type char(1))
begin
  declare current_fy_end date;
  declare sum_actual numeric(12,2);
  declare sum_proj numeric(12,2);
  declare sum_cont numeric(12,2);
  declare begin_date date;

  select CURRENT_FY_END_DATE into current_fy_end from FMS_CFG;

  select DATE_RECEIVED into begin_date from ACCOUNT
   where ACCOUNT.JON=jo_num;

  select sum(ACTUAL_COST) into sum_actual from CONTRACTS
   where CONTRACTS.JON = jo_num
   and CONTRACTS.CONTRACT_TYPE = cont_type
   and CONTRACTS.FY_ENDING >= begin_date
   and CONTRACTS.FY_ENDING <= current_fy_end;

  if (sum_actual is null) then
   set sum_actual = 0.00
  end if;

  select sum(PROJ_COST) into sum_proj from CONTRACTS
   where CONTRACTS.JON = jo_num
   and CONTRACTS.CONTRACT_TYPE = cont_type
   and CONTRACTS.ACTUAL_COST is null
   and CONTRACTS.FY_ENDING >= begin_date
   and CONTRACTS.FY_ENDING <= current_fy_end;

  if (sum_proj is null) then
   set sum_proj = 0.00
  end if;

  set sum_cont = sum_actual + sum_proj;
```

```
 if (cont_type = 'M') then
  update ACCOUNT
   set BAL_CONT_MIPR = INIT_CONT_MIPR - sum_cont
    where ACCOUNT.JON = jo_num
 else
  if (cont_type = 'I') then
   update ACCOUNT
    set BAL_CONT_IPA = INIT_CONT_IPA - sum_cont
     where ACCOUNT.JON = jo_num
  else
   if (cont_type = 'O') then
    update ACCOUNT
     set BAL_CONT_OTH = INIT_CONT_OTH - sum_cont
      where ACCOUNT.JON = jo_num
   end if
  end if
 end if;

end
/
%*********************************************************%



%*********************************************************%
% Procedure CALC_BAL_FAC_LABOR
%*********************************************************%
create procedure %PROC% (IN jo_num char(5))
begin
 declare current_fy_end date;
 declare begin_date date;
 declare sum_chg numeric(12,2);

 select CURRENT_FY_END_DATE into current_fy_end from FMS_CFG;

 select DATE_RECEIVED into begin_date from ACCOUNT
  where ACCOUNT.JON=jo_num;

 select sum(TOTAL_CHG) into sum_chg from LABOR_CHGS, FACULTY
  where FACULTY.EMP_ID_CODE = LABOR_CHGS.EMP_ID_CODE
  and LABOR_CHGS.JON = jo_num
  and LABOR_CHGS.FY_ENDING >= begin_date
```

```
   and LABOR_CHGS.FY_ENDING <= current_fy_end;

  if (sum_chg is null) then
    set sum_chg = 0.00
  end if;

  update ACCOUNT
    set BAL_FAC_LABOR = INIT_FAC_LABOR_$ - sum_chg
      where ACCOUNT.JON = jo_num;
end
/
%************************************************************%



%************************************************************%
% Procedure CALC_BAL_OPTAR
%************************************************************%
create procedure %PROC% (IN jo_num char(5))
begin
  declare current_fy_end date;
  declare sum_actual numeric(12,2);
  declare sum_proj numeric(12,2);
  declare sum_optar numeric(12,2);
  declare begin_date date;

  select CURRENT_FY_END_DATE into current_fy_end from FMS_CFG;

  select DATE_RECEIVED into begin_date from ACCOUNT
    where ACCOUNT.JON=jo_num;

  select sum(ACTUAL_COST) into sum_actual from OPTAR_REQ
    where OPTAR_REQ.JON = jo_num
    and OPTAR_REQ.FY_ENDING >= begin_date
    and OPTAR_REQ.FY_ENDING <= current_fy_end;

  if (sum_actual is null) then
    set sum_actual = 0.00
  end if;

  select sum(PROJ_COST) into sum_proj from OPTAR_REQ
    where OPTAR_REQ.JON = jo_num
```

```
   and OPTAR_REQ.ACTUAL_COST is null
   and OPTAR_REQ.FY_ENDING >= begin_date
   and OPTAR_REQ.FY_ENDING <= current_fy_end;

 if (sum_proj is null) then
   set sum_proj = 0.00
 end if;

 set sum_optar = sum_actual + sum_proj;

 update ACCOUNT
   set BAL_OPTAR = INIT_OPTAR_$ - sum_optar
     where ACCOUNT.JON = jo_num;
end
/
%************************************************************%



%************************************************************%
% Procedure CALC_BAL_SPT_LABOR
%************************************************************%
create procedure %PROC% (IN jo_num char(5))
begin
 declare current_fy_end date;
 declare begin_date date;
 declare sum_chg numeric(12,2);

 select CURRENT_FY_END_DATE into current_fy_end from FMS_CFG;

 select DATE_RECEIVED into begin_date from ACCOUNT
   where ACCOUNT.JON=jo_num;

 select sum(TOTAL_CHG) into sum_chg from LABOR_CHGS, STAFF
   where STAFF.EMP_ID_CODE = LABOR_CHGS.EMP_ID_CODE
   and LABOR_CHGS.JON = jo_num
   and LABOR_CHGS.FY_ENDING >= begin_date
   and LABOR_CHGS.FY_ENDING <= current_fy_end;

 if (sum_chg is null) then
   set sum_chg = 0.00
 end if;
```

```
 update ACCOUNT
   set BAL_SPT_LABOR = INIT_SPT_LABOR_$ - sum_chg
     where ACCOUNT.JON = jo_num;
end
/
%*********************************************************%



%*********************************************************%
% Procedure CALC_BAL_TRAV
%*********************************************************%
create procedure %PROC% (IN jo_num char(5))
begin
 declare current_fy_end date;
 declare sum_actual numeric(12,2);
 declare sum_proj numeric(12,2);
 declare sum_trav numeric(12,2);
 declare begin_date date;

 select CURRENT_FY_END_DATE into current_fy_end from FMS_CFG;

 select DATE_RECEIVED into begin_date from ACCOUNT
  where ACCOUNT.JON=jo_num;

 select sum(ACTUAL_COST) into sum_actual from TRAVEL
  where TRAVEL.JON = jo_num
  and TRAVEL.FY_ENDING >= begin_date
  and TRAVEL.FY_ENDING <= current_fy_end;

 if (sum_actual is null) then
  set sum_actual = 0.00
 end if;

 select sum(PROJ_COST) into sum_proj from TRAVEL
  where TRAVEL.JON = jo_num
  and TRAVEL.ACTUAL_COST is null
  and TRAVEL.FY_ENDING >= begin_date
  and TRAVEL.FY_ENDING <= current_fy_end;

 if (sum_proj is null) then
  set sum_proj = 0.00
```

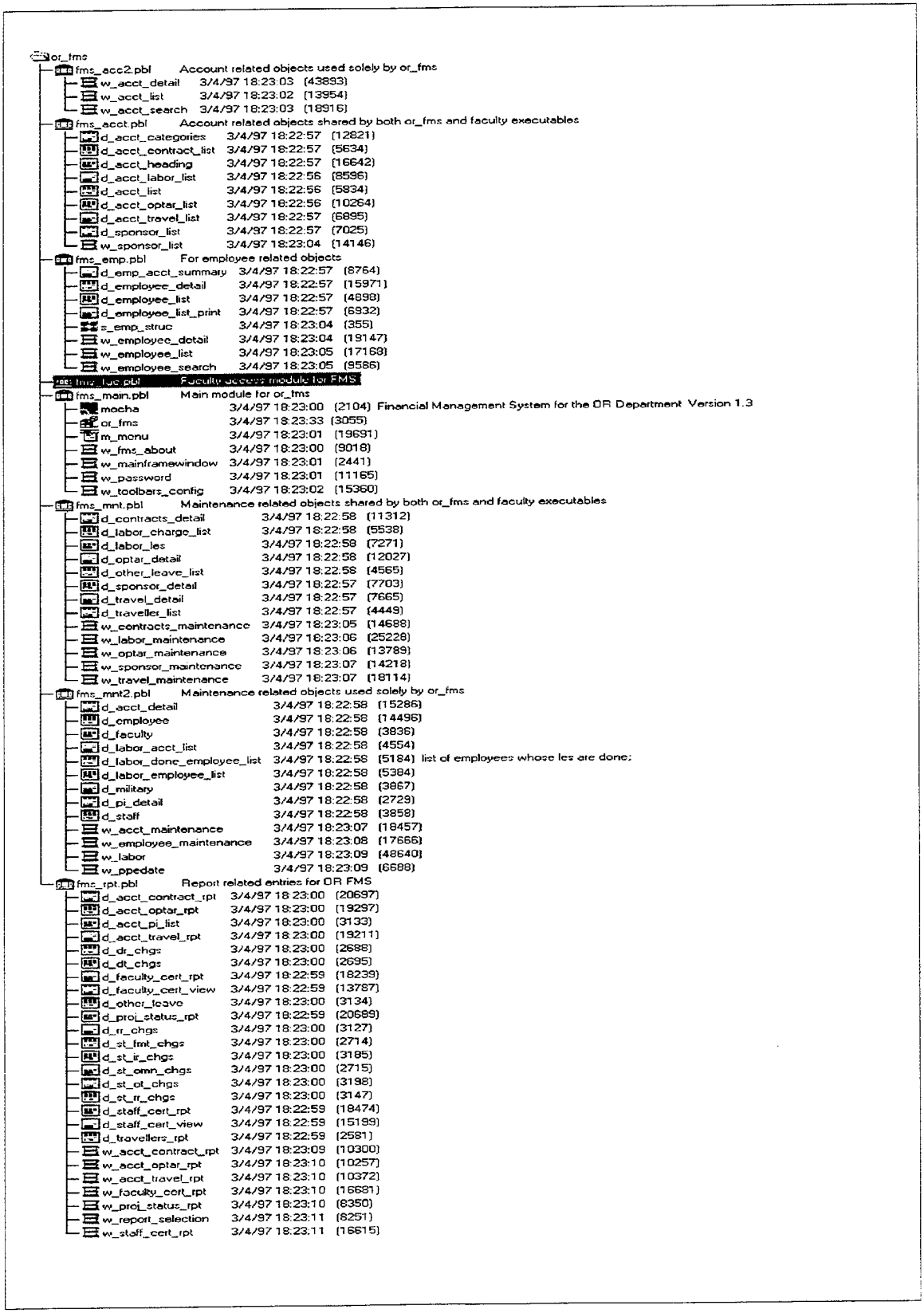
```
  end if;

  set sum_trav = sum_actual + sum_proj;

  update ACCOUNT
    set BAL_TRAVEL = INIT_TRAVEL_$ - sum_trav
      where ACCOUNT.JON = jo_num;
end
/
%*********************************************************%
```

APPENDIX D. FMS *POWERBUILDER* LIBRARY OBJECT LISTING

The FMS PowerBuilder library object listing is shown on the next page.


or_fms
fms_acc2.pbl Account related objects used solely by or_fms
w_acct_detail 3/4/97 18:23:03 (43893)
w_acct_list 3/4/97 18:23:02 (13954)
w_acct_search 3/4/97 18:23:03 (18916)
fms_acct.pbl Account related objects shared by both or_fms and faculty executables
d_acct_categories 3/4/97 18:22:57 (12821)
d_acct_contract_list 3/4/97 18:22:57 (5634)
d_acct_heading 3/4/97 18:22:57 (16642)
d_acct_labor_list 3/4/97 18:22:56 (8596)
d_acct_list 3/4/97 18:22:56 (5834)
d_acct_optar_list 3/4/97 18:22:56 (10264)
d_acct_travel_list 3/4/97 18:22:57 (6895)
d_sponsor_list 3/4/97 18:22:57 (7025)
w_sponsor_list 3/4/97 18:23:04 (14146)
fms_emp.pbl For employee related objects
d_emp_acct_summary 3/4/97 18:22:57 (8764)
d_employee_detail 3/4/97 18:22:57 (15971)
d_employee_list 3/4/97 18:22:57 (4898)
d_employee_list_print 3/4/97 18:22:57 (6932)
s_emp_struc 3/4/97 18:23:04 (355)
w_employee_detail 3/4/97 18:23:04 (19147)
w_employee_list 3/4/97 18:23:05 (17168)
w_employee_search 3/4/97 18:23:05 (9586)
fms_fac.pbl Faculty access module for FMS
fms_main.pbl Main module for or_fms
mocha 3/4/97 18:23:00 (2104) Financial Management System for the OR Department Version 1.3
or_fms 3/4/97 18:23:33 (3055)
m_menu 3/4/97 18:23:01 (19691)
w_fms_about 3/4/97 18:23:00 (9018)
w_mainframewindow 3/4/97 18:23:01 (2441)
w_password 3/4/97 18:23:01 (11165)
w_toolbars_config 3/4/97 18:23:02 (15360)
fms_mnt.pbl Maintenance related objects shared by both or_fms and faculty executables
d_contracts_detail 3/4/97 18:22:58 (11312)
d_labor_charge_list 3/4/97 18:22:58 (5538)
d_labor_les 3/4/97 18:22:58 (7271)
d_optar_detail 3/4/97 18:22:58 (12027)
d_other_leave_list 3/4/97 18:22:58 (4565)
d_sponsor_detail 3/4/97 18:22:57 (7703)
d_travel_detail 3/4/97 18:22:57 (7665)
d_traveller_list 3/4/97 18:22:57 (4449)
w_contracts_maintenance 3/4/97 18:23:05 (14688)
w_labor_maintenance 3/4/97 18:23:06 (25228)
w_optar_maintenance 3/4/97 18:23:06 (13789)
w_sponsor_maintenance 3/4/97 18:23:07 (14218)
w_travel_maintenance 3/4/97 18:23:07 (18114)
fms_mnt2.pbl Maintenance related objects used solely by or_fms
d_acct_detail 3/4/97 18:22:58 (15286)
d_employee 3/4/97 18:22:58 (14496)
d_faculty 3/4/97 18:22:58 (3836)
d_labor_acct_list 3/4/97 18:22:58 (4554)
d_labor_done_employee_list 3/4/97 18:22:58 (5184) list of employees whose les are done:
d_labor_employee_list 3/4/97 18:22:58 (5384)
d_military 3/4/97 18:22:58 (3867)
d_pi_detail 3/4/97 18:22:58 (2729)
d_staff 3/4/97 18:22:58 (3858)
w_acct_maintenance 3/4/97 18:23:07 (18457)
w_employee_maintenance 3/4/97 18:23:08 (17666)
w_labor 3/4/97 18:23:09 (48640)
w_ppedate 3/4/97 18:23:09 (6688)
fms_rpt.pbl Report related entries for OR FMS
d_acct_contract_rpt 3/4/97 18:23:00 (20697)
d_acct_optar_rpt 3/4/97 18:23:00 (19297)
d_acct_pi_list 3/4/97 18:23:00 (3133)
d_acct_travel_rpt 3/4/97 18:23:00 (19211)
d_dr_chgs 3/4/97 18:23:00 (2688)
d_dt_chgs 3/4/97 18:23:00 (2695)
d_faculty_cert_rpt 3/4/97 18:22:59 (18239)
d_faculty_cert_view 3/4/97 18:22:59 (13787)
d_other_leave 3/4/97 18:23:00 (3134)
d_proj_status_rpt 3/4/97 18:22:59 (20689)
d_rr_chgs 3/4/97 18:23:00 (3127)
d_st_fmt_chgs 3/4/97 18:23:00 (2714)
d_st_ir_chgs 3/4/97 18:23:00 (3185)
d_st_omn_chgs 3/4/97 18:23:00 (2715)
d_st_ot_chgs 3/4/97 18:23:00 (3198)
d_st_rr_chgs 3/4/97 18:23:00 (3147)
d_staff_cert_rpt 3/4/97 18:22:59 (18474)
d_staff_cert_view 3/4/97 18:22:59 (15199)
d_travellers_rpt 3/4/97 18:22:59 (2581)
w_acct_contract_rpt 3/4/97 18:23:09 (10300)
w_acct_optar_rpt 3/4/97 18:23:10 (10257)
w_acct_travel_rpt 3/4/97 18:23:10 (10372)
w_faculty_cert_rpt 3/4/97 18:23:10 (16681)
w_proj_status_rpt 3/4/97 18:23:10 (8350)
w_report_selection 3/4/97 18:23:11 (8251)
w_staff_cert_rpt 3/4/97 18:23:11 (16615)

APPENDIX E. FMS *APPMODELER* REPORT

The partial *AppModeler* report produced from the FMS physical data model begins on the next page.

Full PDM report

Model Information

Project Name:	fms
Project Code:	FMS
Database:	Watcom SQL 4.0
Name:	fms
Code:	FMS
Label:	Ops Research Dept Financial Management System
Author:	Alan E. Pires
Version:	1.01
Created On:	11/30/95 8:01 AM **Modified On:** 2/3/97 4:51 PM

Model Description

Financial Management System for the Operations Research Department

Begin Script

End Script

Business Rules

Domains

Tables

Table List

Name	Code	Number
account	ACCOUNT	0
adp_proj_info	ADP_PROJ_INFO	0
contracts	CONTRACTS	0
department	DEPARTMENT	0

employee	EMPLOYEE	0
faculty	FACULTY	0
fms_cfg	FMS_CFG	0
labor_chgs	LABOR_CHGS	0
labor_les	LABOR_LES	0
military	MILITARY	0
optar_req	OPTAR_REQ	0
other_leave	OTHER_LEAVE	0
other_lv_type	OTHER_LV_TYPE	0
pi	PI	0
salary_history	SALARY_HISTORY	0
sponsor	SPONSOR	0
staff	STAFF	0
travel	TRAVEL	0
travel_requests	TRAVEL_REQUESTS	0

Table account

Name:	account
Code:	ACCOUNT
Label:	Account Information
Number:	
PK constraint:	

Options

Column List

Name	Code	Type	P	M
bal_cont_ipa	BAL_CONT_IPA	decimal(12,2)	No	No
bal_cont_mipr	BAL_CONT_MIPR	decimal(12,2)	No	No
bal_cont_oth	BAL_CONT_OTH	decimal(12,2)	No	No

bal_fac_labor	BAL_FAC_LABOR	decimal(12,2)	No	No
bal_optar	BAL_OPTAR	decimal(12,2)	No	No
bal_spt_labor	BAL_SPT_LABOR	decimal(12,2)	No	No
bal_travel	BAL_TRAVEL	decimal(12,2)	No	No
budget_page_date	BUDGET_PAGE_DATE	date	No	No
date_received	DATE_RECEIVED	date	No	No
expir_date	EXPIR_DATE	date	No	No
fund_type	FUND_TYPE	char(2)	No	Yes
indirect_cost	INDIRECT_COST	decimal(12,2)	No	Yes
init_cont_ipa	INIT_CONT_IPA	decimal(12,2)	No	Yes
init_cont_mipr	INIT_CONT_MIPR	decimal(12,2)	No	Yes
init_cont_oth	INIT_CONT_OTH	decimal(12,2)	No	Yes
init_fac_labor_$	INIT_FAC_LABOR_$	decimal(12,2)	No	Yes
init_optar_$	INIT_OPTAR_$	decimal(12,2)	No	Yes
init_spt_labor_$	INIT_SPT_LABOR_$	decimal(12,2)	No	Yes
init_travel_$	INIT_TRAVEL_$	decimal(12,2)	No	Yes
jon	JON	char(5)	Yes	Yes
labor_jon	LABOR_JON	char(5)	No	No
remarks	REMARKS	char(100)	No	No
segment_#s	SEGMENT_#S	char(9)	No	No
serial_#s	SERIAL_#S	char(11)	No	No
spon_id_code	SPON_ID_CODE	char(6)	No	No
title	TITLE	char(40)	No	No

BAL_CONT_IPA

Check

Domain:					
Low value:					
High value:					
Default value:					
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

BAL_CONT_MIPR

Check

Domain:					
Low value:					
High value:					
Default value:					
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

BAL_CONT_OTH

Check

Domain:					
Low value:					
High value:					
Default value:					
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

BAL_FAC_LABOR

Check

Domain:
Low value:
High value:
Default value:
Unit:
Format:
Uppercase: No **Lowercase:** No **Can't modify:** No
List of values:

BAL_OPTAR

Check

Domain:
Low value:
High value:
Default value:
Unit:
Format:
Uppercase: No **Lowercase:** No **Can't modify:** No
List of values:

BAL_SPT_LABOR

Check

Domain:
Low value:
High value:
Default value:
Unit:
Format:

Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

BAL_TRAVEL

Check

Domain:					
Low value:					
High value:					
Default value:					
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

BUDGET_PAGE_DATE

Check

Domain:					
Low value:					
High value:					
Default value:					
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

DATE_RECEIVED

Check

Domain:	
Low value:	

High value:					
Default value:					
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

EXPIR_DATE

Check

Domain:					
Low value:					
High value:					
Default value:					
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

FUND_TYPE

Check

Domain:					
Low value:					
High value:					
Default value:					
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

INDIRECT_COST

Check

Domain:					
Low value:	0.00				
High value:					
Default value:	0.00				
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

INIT_CONT_IPA

Check

Domain:					
Low value:	0				
High value:					
Default value:	0				
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

INIT_CONT_MIPR

Check

Domain:					
Low value:	0				
High value:					
Default value:	0				
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

INIT_CONT_OTH

Check

Domain:					
Low value:	0				
High value:					
Default value:	0				
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

INIT_FAC_LABOR_$

Check

Domain:					
Low value:	0.00				
High value:					
Default value:	0.00				
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

INIT_OPTAR_$

Check

Domain:	
Low value:	0.00
High value:	
Default value:	0.00
Unit:	

Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

INIT_SPT_LABOR_$

Check

Domain:					
Low value:	0.00				
High value:					
Default value:	0.00				
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

INIT_TRAVEL_$

Check

Domain:					
Low value:	0.00				
High value:					
Default value:	0.00				
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

JON

Check

Domain:	

Low value:					
High value:					
Default value:					
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

LABOR_JON

Check

Domain:					
Low value:					
High value:					
Default value:					
Unit:					
Format:					
Uppercase:	Yes	**Lowercase:**	No	**Can't modify:**	No
List of values:					

REMARKS

Check

Domain:					
Low value:					
High value:					
Default value:					
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

SEGMENT_#S

Check

Domain:
Low value:
High value:
Default value:
Unit:
Format:
Uppercase: No **Lowercase:** No **Can't modify:** No
List of values:

SERIAL_#S

Check

Domain:
Low value:
High value:
Default value:
Unit:
Format:
Uppercase: No **Lowercase:** No **Can't modify:** No
List of values:

SPON_ID_CODE

Check

Domain:
Low value:
High value:
Default value:
Unit:
Format:
Uppercase: No **Lowercase:** No **Can't modify:** No
List of values:

TITLE

Check

Domain:
Low value:
High value:
Default value:
Unit:
Format:
Uppercase: No **Lowercase:** No **Can't modify:** No
List of values:

Index List

Index Code	P	F	U	C	Column Code	Sort
ACCOUNT_FK1	No	Yes	No	No	SPON_ID_CODE	ASC
ACCOUNT_PK	Yes	No	Yes	No	JON	ASC

Reference to List

Reference to	Primary Key	Foreign Key
SPONSOR	SPON_ID_CODE	SPON_ID_CODE

Reference by List

Referenced by	Primary Key	Foreign Key
TRAVEL	JON	JON
LABOR_CHGS	JON	JON
CONTRACTS	JON	JON
OPTAR_REQ	JON	JON
PI	JON	JON

Table adp_proj_info

Name: adp_proj_info
Code: ADP_PROJ_INFO
Label: ADP Project Information
Number:
PK constraint:

Options

Column List

Name	Code	Type	P	M
adp_proj_#	ADP_PROJ_#	char(7)	Yes	Yes
dept_code	DEPT_CODE	char(2)	No	No
fy_ending	FY_ENDING	date	No	No
poc_code	POC_CODE	char(4)	No	No
proj_cost_auth	PROJ_COST_AUTH	decimal(12,2)	No	No
proj_mgr_code	PROJ_MGR_CODE	char(4)	No	No
proj_name	PROJ_NAME	char(40)	No	No

ADP_PROJ_#

Check

Domain:
Low value:
High value:
Default value:
Unit:
Format:
Uppercase: No **Lowercase:** No **Can't modify:** No
List of values:

DEPT_CODE

Check

Domain:					
Low value:					
High value:					
Default value:					
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

FY_ENDING

Check

Domain:					
Low value:					
High value:					
Default value:	09/30/97				
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

POC_CODE

Check

Domain:					
Low value:					
High value:					
Default value:					
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

PROJ_COST_AUTH

Check

Domain:
Low value:
High value:
Default value:
Unit:
Format:
Uppercase: No **Lowercase:** No **Can't modify:** No
List of values:

PROJ_MGR_CODE

Check

Domain:
Low value:
High value:
Default value:
Unit:
Format:
Uppercase: No **Lowercase:** No **Can't modify:** No
List of values:

PROJ_NAME

Check

Domain:
Low value:
High value:
Default value:
Unit:

Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

Index List

Index Code	P	F	U	C	Column Code	Sort
ADP_PROJ_INFO_FK1	No	Yes	No	No	DEPT_CODE	ASC
ADP_PROJ_INFO_FK2	No	Yes	No	No	PROJ_MGR_CODE	ASC
ADP_PROJ_INFO_FK3	No	Yes	No	No	POC_CODE	ASC
ADP_PROJ_INFO_PK	Yes	No	Yes	No	ADP_PROJ_#	ASC

Reference to List

Reference to	Primary Key	Foreign Key
DEPARTMENT	DEPT_CODE	DEPT_CODE
EMPLOYEE	EMP_ID_CODE	PROJ_MGR_CODE
EMPLOYEE	EMP_ID_CODE	POC_CODE

Reference by List

Referenced by	Primary Key	Foreign Key
OPTAR_REQ	ADP_PROJ_#	ADP_PROJ_#

Table contracts

Name:	contracts
Code:	CONTRACTS
Label:	Departmental Contracts (charged to departmental accounts)
Number:	
PK constraint:	

Options

Description

Departmental Contracts (charged to departmental accounts)

Column List

Name	Code	Type	P	M
actual_cost	ACTUAL_COST	decimal(12,2)	No	No
contract_type	CONTRACT_TYPE	char(1)	Yes	Yes
contractor	CONTRACTOR	char(20)	No	No
delivery_date	DELIVERY_DATE	date	No	No
description	DESCRIPTION	char(50)	No	No
doc_#	DOC_#	char(9)	Yes	Yes
fy_ending	FY_ENDING	date	No	Yes
jon	JON	char(5)	Yes	Yes
order_date	ORDER_DATE	date	No	No
po_#	PO_#	char(12)	No	No
po_date	PO_DATE	date	No	No
proj_cost	PROJ_COST	decimal(12,2)	No	No
requester	REQUESTER	char(4)	Yes	Yes

ACTUAL_COST

Check

Domain:
Low value:
High value:
Default value:
Unit:
Format:
Uppercase: No **Lowercase:** No **Can't modify:** No
List of values:

CONTRACT_TYPE

Check

Domain:
Low value:
High value:
Default value:
Unit:
Format:
Uppercase: No **Lowercase:** No **Can't modify:** No
List of values:

CONTRACTOR

Check

Domain:
Low value:
High value:
Default value:
Unit:
Format:
Uppercase: No **Lowercase:** No **Can't modify:** No
List of values:

DELIVERY_DATE

Check

Domain:
Low value:
High value:
Default value:
Unit:
Format:
Uppercase: No **Lowercase:** No **Can't modify:** No
List of values:

DESCRIPTION

Check

Domain:					
Low value:					
High value:					
Default value:					
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

DOC_#

Check

Domain:					
Low value:					
High value:					
Default value:					
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

FY_ENDING

Check

Domain:	
Low value:	
High value:	
Default value:	9/30/97
Unit:	

Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

JON

Check

Domain:					
Low value:					
High value:					
Default value:					
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

ORDER_DATE

Check

Domain:					
Low value:					
High value:					
Default value:					
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

PO_#

Check

Domain:	

Low value:					
High value:					
Default value:					
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

PO_DATE

Check

Domain:					
Low value:					
High value:					
Default value:					
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

PROJ_COST

Check

Domain:					
Low value:					
High value:					
Default value:					
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

REQUESTER

Check

Domain:
Low value:
High value:
Default value:
Unit:
Format:
Uppercase: No **Lowercase:** No **Can't modify:** No
List of values:

Index List

Index Code	P	F	U	C	Column Code	Sort
CONTRACTS_PK	Yes	No	Yes	No	JON CONTRACT_TYPE REQUESTER DOC_#	ASC ASC ASC ASC

Reference to List

Reference to	Primary Key	Foreign Key
ACCOUNT	JON	JON
EMPLOYEE	EMP_ID_CODE	REQUESTER

Table department

Name: department
Code: DEPARTMENT
Label: Department Info
Number:
PK constraint:

Options

Column List

Name	Code	Type	P	M
chair_code	CHAIR_CODE	char(4)	No	No
dept_code	DEPT_CODE	char(2)	Yes	Yes
dept_name	DEPT_NAME	char(40)	No	No

CHAIR_CODE

Check

Domain:
Low value:
High value:
Default value:
Unit:
Format:
Uppercase: No **Lowercase:** No **Can't modify:** No
List of values:

DEPT_CODE

Check

Domain:
Low value:
High value:
Default value:
Unit:
Format:
Uppercase: No **Lowercase:** No **Can't modify:** No
List of values:

DEPT_NAME

Check

Domain:
Low value:
High value:
Default value:
Unit:
Format:
Uppercase: No **Lowercase:** No **Can't modify:** No
List of values:

Index List

Index Code	P	F	U	C	Column Code	Sort
DEPARTMENT_PK	Yes	No	Yes	No	DEPT_CODE	ASC

Reference by List

Referenced by	Primary Key	Foreign Key
EMPLOYEE	DEPT_CODE	DEPT_CODE
ADP_PROJ_INFO	DEPT_CODE	DEPT_CODE

Table employee

Name: employee
Code: EMPLOYEE
Label: Employee Information
Number:
PK constraint:

Options

Column List

Name	Code	Type	P	M
accel_rate	ACCEL_RATE	decimal(3, 2)	No	No
base_salary	BASE_SALARY	decimal(10 ,2)	No	No
bldg_#	BLDG_#	char(3)	No	No
category	CATEGORY	char	No	Yes
city	CITY	char(15)	No	No
dept_code	DEPT_CODE	char(2)	No	No
eff_sal_date	EFF_SAL_DATE	date	No	No
emp_code	EMP_CODE	char(2)	No	Yes
emp_id_code	EMP_ID_CODE	char(4)	Yes	Yes
first name	FIRST_NAME	char(15)	No	No
home_phone	HOME_PHONE	char(13)	No	No
last_name	LAST_NAME	char(15)	No	Yes
mi	MI	char(1)	No	No
room_#	ROOM_#	char(5)	No	No
spouse_fname	SPOUSE_FNAME	char(15)	No	No
ssn	SSN	char(11)	No	No
state	STATE	char(2)	No	No
street address	STREET_ADDRESS	char(20)	No	No
term_date	TERM_DATE	date	No	No
work_phone	WORK_PHONE	char(13)	No	No
zipcode	ZIPCODE	char(10)	No	No

ACCEL_RATE

Check

Domain:
Low value:
High value:
Default value: 1.43
Unit:
Format:
Uppercase: No **Lowercase:** No **Can't modify:** No
List of values:

BASE_SALARY

Check

Domain:
Low value:
High value:
Default value:
Unit:
Format:
Uppercase: No **Lowercase:** No **Can't modify:** No
List of values:

BLDG_#

Check

Domain:
Low value:
High value:
Default value:
Unit:
Format:
Uppercase: No **Lowercase:** No **Can't modify:** No
List of values:

CATEGORY

Check

Domain:
Low value:
High value:
Default value:
Unit:
Format:

Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

CITY

Check

Domain:					
Low value:					
High value:					
Default value:					
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

DEPT_CODE

Check

Domain:					
Low value:					
High value:					
Default value:					
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

EFF_SAL_DATE

Check

Domain:	
Low value:	

High value:					
Default value:	10/01/95				
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

EMP_CODE

Check

Domain:					
Low value:					
High value:					
Default value:					
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

EMP_ID_CODE

Check

Domain:					
Low value:					
High value:					
Default value:					
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

FIRST_NAME

Check

Domain:					
Low value:					
High value:					
Default value:					
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

HOME_PHONE

Check

Domain:					
Low value:					
High value:					
Default value:					
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

LAST_NAME

Check

Domain:					
Low value:					
High value:					
Default value:					
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

MI

Check

Domain:					
Low value:					
High value:					
Default value:					
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

ROOM_#

Check

Domain:					
Low value:					
High value:					
Default value:					
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

SPOUSE_FNAME

Check

Domain:	
Low value:	
High value:	
Default value:	
Unit:	

Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

SSN

Check

Domain:					
Low value:					
High value:					
Default value:					
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

STATE

Check

Domain:					
Low value:					
High value:					
Default value:					
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

STREET_ADDRESS

Check

Domain:	

Low value:					
High value:					
Default value:					
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

TERM_DATE

Check

Domain:					
Low value:					
High value:					
Default value:					
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

WORK_PHONE

Check

Domain:					
Low value:					
High value:					
Default value:					
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

ZIPCODE

Check

Domain:
Low value:
High value:
Default value:
Unit:
Format:
Uppercase: No **Lowercase:** No **Can't modify:** No
List of values:

Index List

Index Code	P	F	U	C	Column Code	Sort
EMPLOYEE_FK1	No	Yes	No	No	DEPT_CODE	ASC
EMPLOYEE_PK	Yes	Yes	Yes	No	EMP_ID_CODE	ASC

Reference to List

Reference to	Primary Key	Foreign Key
DEPARTMENT	DEPT_CODE	DEPT_CODE

Reference by List

Referenced by	Primary Key	Foreign Key
LABOR_CHGS	EMP_ID_CODE	EMP_ID_CODE
OPTAR_REQ	EMP_ID_CODE	EMP_ID_CODE
SALARY_HISTORY	EMP_ID_CODE	EMP_ID_CODE
PI	EMP_ID_CODE	EMP_ID_CODE
CONTRACTS	EMP_ID_CODE	REQUESTER
LABOR_LES	EMP_ID_CODE	EMP_ID_CODE
FACULTY	EMP_ID_CODE	EMP_ID_CODE
STAFF	EMP_ID_CODE	EMP_ID_CODE
MILITARY	EMP_ID_CODE	EMP_ID_CODE
ADP_PROJ_INFO	EMP_ID_CODE	PROJ_MGR_CODE
ADP_PROJ_INFO	EMP_ID_CODE	POC_CODE

Table faculty

Name:	faculty
Code:	FACULTY
Label:	Faculty Specialization of Employee Table
Number:	
PK constraint:	

Options

Column List

Name	Code	Type	P	M
civ_grade	CIV_GRADE	char(5)	No	No
emp_id_code	EMP_ID_CODE	char(4)	Yes	Yes
step	STEP	char(2)	No	No

CIV_GRADE

Check

Domain:					
Low value:					
High value:					
Default value:					
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

EMP_ID_CODE

Check

Domain:
Low value:
High value:
Default value:
Unit:
Format:
Uppercase: No **Lowercase:** No **Can't modify:** No
List of values:

STEP

Check

Domain:
Low value:
High value:
Default value:
Unit:
Format:
Uppercase: No **Lowercase:** No **Can't modify:** No
List of values:

Index List

Index Code	P	F	U	C	Column Code	Sort
FACULTY_PK	Yes	Yes	Yes	No	EMP_ID_CODE	ASC

Reference to List

Reference to	Primary Key	Foreign Key
EMPLOYEE	EMP_ID_CODE	EMP_ID_CODE

Table fms_cfg

Name: fms_cfg
Code: FMS_CFG

Label: FMS Configuration Info
Number:
PK constraint:

Options

Column List

Name	Code	Type	P	M
current_fy_end_date	CURRENT_FY_END_DATE	date	Yes	Yes
ot_cap	OT_CAP	decimal(10,2)	No	No
rr_ot_rate_fact	RR_OT_RATE_FACT	decimal(6,4)	No	No
yr_labor_hrs	YR_LABOR_HRS	integer	No	No

CURRENT_FY_END_DATE

Check

Domain:
Low value:
High value:
Default value:
Unit:
Format:
Uppercase: No **Lowercase:** No **Can't modify:** No
List of values:

OT_CAP

Check

Domain:
Low value:

High value:					
Default value:					
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

RR_OT_RATE_FACT

Check

Domain:					
Low value:					
High value:					
Default value:					
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

YR_LABOR_HRS

Check

Domain:					
Low value:					
High value:					
Default value:					
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

Index List

Index Code	P	F	U	C	Column Code	Sort
FMS_CFG_PK	Yes	No	Yes	No	CURRENT_FY_END_DATE	ASC

Table labor_chgs

Name: labor_chgs
Code: LABOR_CHGS
Label: Labor charges made against accounts
Number:
PK constraint:

Options

Description

This table contains the labor charges made against accounts by pay period ending date and employee.

Column List

Name	Code	Type	P	M
emp_id_code	EMP_ID_CODE	char(4)	Yes	Yes
fy_ending	FY_ENDING	date	No	Yes
hours	HOURS	integer	No	Yes
jon	JON	char(5)	Yes	Yes
ot_hours	OT_HOURS	integer	No	Yes
ppe_date	PPE_DATE	date	Yes	Yes
total_chg	TOTAL_CHG	decimal(12,2)	No	No

EMP_ID_CODE

Check

Domain:
Low value:
High value:

Default value:					
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

FY_ENDING

Check

Domain:					
Low value:					
High value:					
Default value:	9/30/97				
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

HOURS

Check

Domain:					
Low value:	0				
High value:					
Default value:					
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

JON

Check

Domain:					
Low value:					
High value:					
Default value:					
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

OT_HOURS

Check

Domain:					
Low value:	0				
High value:					
Default value:	0				
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

PPE_DATE

Check

Domain:					
Low value:					
High value:					
Default value:					
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

TOTAL_CHG

Check

Domain:
Low value:
High value:
Default value:
Unit:
Format:
Uppercase: No **Lowercase:** No **Can't modify:** No
List of values:

Index List

Index Code	P	F	U	C	Column Code	Sort
LABOR_CHGS_PK	Yes	Yes	Yes	No	EMP_ID_CODE PPE_DATE JON	ASC ASC ASC

Reference to List

Reference to	Primary Key	Foreign Key
LABOR_LES	EMP_ID_CODE PPE_DATE	EMP_ID_CODE PPE_DATE
ACCOUNT	JON	JON
EMPLOYEE	EMP_ID_CODE	EMP_ID_CODE

Table labor_les

Name: labor_les
Code: LABOR_LES
Label: Labor -- Leave and Holiday Charges
Number:
PK constraint:

Options

Column List

Name	Code	Type	P	M
al_hours	AL_HOURS	integer	No	No
emp_id_code	EMP_ID_CODE	char(4)	Yes	Yes
hol_hours	HOL_HOURS	integer	No	No
lwop_hours	LWOP_HOURS	integer	No	No
ppe_date	PPE_DATE	date	Yes	Yes
sl_hours	SL_HOURS	integer	No	No

AL_HOURS

Check

Domain:
Low value:
High value:
Default value:
Unit:
Format:
Uppercase: No **Lowercase:** No **Can't modify:** No
List of values:

EMP_ID_CODE

Check

Domain:
Low value:
High value:
Default value:
Unit:
Format:

Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

HOL_HOURS

Check

Domain:					
Low value:					
High value:					
Default value:					
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

LWOP_HOURS

Check

Domain:					
Low value:					
High value:					
Default value:					
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

PPE_DATE

Check

Domain:	
Low value:	

High value:					
Default value:					
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

SL_HOURS

Check

Domain:					
Low value:					
High value:					
Default value:					
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

Index List

Index Code	P	F	U	C	Column Code	Sort
LABOR_LES_PK	Yes	No	Yes	No	EMP_ID_CODE PPE_DATE	ASC ASC

Reference to List

Reference to	Primary Key	Foreign Key
EMPLOYEE	EMP_ID_CODE	EMP_ID_CODE

Reference by List

Referenced by	Primary Key	Foreign Key
LABOR_CHGS	EMP_ID_CODE PPE_DATE	EMP_ID_CODE PPE_DATE

OTHER_LEAVE	EMP_ID_CODE PPE_DATE	EMP_ID_CODE PPE_DATE

Table military

Name: military
Code: MILITARY
Label: Military Specialization of Employee Table
Number:
PK constraint:

Options

Column List

Name	Code	Type	P	M
emp_id_code	EMP_ID_CODE	char(4)	Yes	Yes
mil_grade	MIL_GRADE	char(5)	No	No
service	SERVICE	char(4)	No	No

EMP_ID_CODE

Check

Domain:
Low value:
High value:
Default value:
Unit:
Format:
Uppercase: No **Lowercase:** No **Can't modify:** No
List of values:

MIL_GRADE

Check

Domain:
Low value:
High value:
Default value:
Unit:
Format:
Uppercase: No **Lowercase:** No **Can't modify:** No
List of values:

SERVICE

Check

Domain:
Low value:
High value:
Default value:
Unit:
Format:
Uppercase: No **Lowercase:** No **Can't modify:** No
List of values:

Index List

Index Code	P	F	U	C	Column Code	Sort
MILITARY_PK	Yes	Yes	Yes	No	EMP_ID_CODE	ASC

Reference to List

Reference to	Primary Key	Foreign Key
EMPLOYEE	EMP_ID_CODE	EMP_ID_CODE

Table optar_req

Name:	optar_req
Code:	OPTAR_REQ
Label:	OPTAR Request Information
Number:	
PK constraint:	

Options

Description

OPTAR Request Information

Column List

Name	Code	Type	P	M
actual_cost	ACTUAL_COST	decimal(11,2)	No	No
adp_proj_#	ADP_PROJ_#	char(7)	No	No
category	CATEGORY	char(1)	No	No
description	DESCRIPTION	char(50)	No	No
doc_#	DOC_#	char(9)	Yes	Yes
emp_id_code	EMP_ID_CODE	char(4)	Yes	Yes
fy_ending	FY_ENDING	date	No	Yes
issued_by	ISSUED_BY	char(15)	No	No
jon	JON	char(5)	Yes	Yes
order_date	ORDER_DATE	date	No	No
po_#	PO_#	char(12)	No	No
po_date	PO_DATE	date	No	No
proj_cost	PROJ_COST	decimal(11,2)	No	No
recvd_date	RECVD_DATE	date	No	No

ACTUAL_COST

Check

Domain:					
Low value:					
High value:					
Default value:					
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

ADP_PROJ_#

Check

Domain:					
Low value:					
High value:					
Default value:					
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

CATEGORY

Check

Domain:					
Low value:					
High value:					
Default value:					
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

DESCRIPTION

Check

Domain:					
Low value:					
High value:					
Default value:					
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

DOC_#

Check

Domain:					
Low value:					
High value:					
Default value:					
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

EMP_ID_CODE

Check

Domain:	
Low value:	
High value:	
Default value:	
Unit:	
Format:	

Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

FY_ENDING

Check

Domain:					
Low value:					
High value:					
Default value:	9/30/97				
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

ISSUED_BY

Check

Domain:					
Low value:					
High value:					
Default value:					
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

JON

Check

Domain:	
Low value:	

High value:					
Default value:					
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

ORDER_DATE

Check

Domain:					
Low value:					
High value:					
Default value:					
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

PO_#

Check

Domain:					
Low value:					
High value:					
Default value:					
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

PO_DATE

Check

Domain:					
Low value:					
High value:					
Default value:					
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

PROJ_COST

Check

Domain:					
Low value:					
High value:					
Default value:					
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

RECVD_DATE

Check

Domain:					
Low value:					
High value:					
Default value:					
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

Index List

Index Code	P	F	U	C	Column Code	Sort
OPTAR_REQ_FK1	No	Yes	No	No	ADP_PROJ_#	ASC
OPTAR_REQ_PK	Yes	No	Yes	No	JON	ASC
					EMP_ID_CODE	ASC
					DOC_#	ASC

Reference to List

Reference to	Primary Key	Foreign Key
EMPLOYEE	EMP_ID_CODE	EMP_ID_CODE
ACCOUNT	JON	JON
ADP_PROJ_INFO	ADP_PROJ_#	ADP_PROJ_#

Table other_leave

Name: other_leave
Code: OTHER_LEAVE
Label: "Other" leave info per employee per pay period
Number:
PK constraint:

Options

Description

"Other" leave info per employee per pay period

Column List

Name	Code	Type	P	M
emp_id_code	EMP_ID_CODE	char(4)	Yes	Yes
hours	HOURS	integer	No	No

ppe_date	PPE_DATE	date	Yes	Yes
type	TYPE	char(2)	Yes	Yes

EMP_ID_CODE

Check

Domain:
Low value:
High value:
Default value:
Unit:
Format:
Uppercase: No **Lowercase:** No **Can't modify:** No
List of values:

HOURS

Check

Domain:
Low value:
High value:
Default value:
Unit:
Format:
Uppercase: No **Lowercase:** No **Can't modify:** No
List of values:

PPE_DATE

Check

Domain:
Low value:
High value:

Default value:					
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

TYPE

Check

Domain:					
Low value:					
High value:					
Default value:					
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

Index List

Index Code	P	F	U	C	Column Code	Sort
OTHER_LEAVE_PK	Yes	Yes	Yes	No	EMP_ID_CODE PPE_DATE TYPE	ASC ASC ASC

Reference to List

Reference to	Primary Key	Foreign Key
LABOR_LES	EMP_ID_CODE PPE_DATE	EMP_ID_CODE PPE_DATE
OTHER_LV_TYPE	OTHER_LV_TYPE_CODE	TYPE

Table other_lv_type

Name:	other_lv_type

Code:	OTHER_LV_TYPE
Label:	Other Leave Type Lookup Table
Number:	
PK constraint:	

Options

Column List

Name	Code	Type	P	M
description	DESCRIPTION	char(25)	No	No
other_lv_type_code	OTHER_LV_TYPE_CODE	char(2)	Yes	Yes

DESCRIPTION

Check

Domain:					
Low value:					
High value:					
Default value:					
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

OTHER_LV_TYPE_CODE

Check

Domain:	
Low value:	
High value:	
Default value:	
Unit:	

Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

Index List

Index Code	P	F	U	C	Column Code	Sort
OTHER_LV_TYPE_PK	Yes	No	Yes	No	OTHER_LV_TYPE_CODE	ASC

Reference by List

Referenced by	Primary Key	Foreign Key
OTHER_LEAVE	OTHER_LV_TYPE_CODE	TYPE

Table pi

Name:	pi
Code:	PI
Label:	Principal Investigator
Number:	
PK constraint:	

Options

Column List

Name	Code	Type	P	M
emp_id_code	EMP_ID_CODE	char(4)	Yes	Yes
jon	JON	char(5)	Yes	Yes

EMP_ID_CODE

Check

Domain:					
Low value:					
High value:					
Default value:					
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

JON

Check

Domain:					
Low value:					
High value:					
Default value:					
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

Index List

Index Code	P	F	U	C	Column Code	Sort
PI_PK	Yes	Yes	Yes	No	EMP_ID_CODE JON	ASC ASC

Reference to List

Reference to	Primary Key	Foreign Key
EMPLOYEE	EMP_ID_CODE	EMP_ID_CODE
ACCOUNT	JON	JON

Table salary_history

Name: salary_history
Code: SALARY_HISTORY
Label: Employee salary history (including acceleration rate)
Number:
PK constraint:

Options

Description

Employee salary history (including acceleration rate)

Column List

Name	Code	Type	P	M
accel_rate	ACCEL_RATE	decimal(3, 2)	No	Yes
base_salary	BASE_SALARY	decimal(10 ,2)	No	Yes
begin_date	BEGIN_DATE	date	Yes	Yes
emp_id_code	EMP_ID_CODE	char(4)	Yes	Yes
end_date	END_DATE	date	No	Yes

ACCEL_RATE

Check

Domain:
Low value:
High value:
Default value:
Unit:
Format:
Uppercase: No **Lowercase:** No **Can't modify:** No
List of values:

BASE_SALARY

Check

Domain:					
Low value:					
High value:					
Default value:					
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

BEGIN_DATE

Check

Domain:					
Low value:					
High value:					
Default value:					
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

EMP_ID_CODE

Check

Domain:	
Low value:	
High value:	
Default value:	
Unit:	
Format:	

Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

END_DATE

Check

Domain:					
Low value:					
High value:					
Default value:					
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

Index List

Index Code	P	F	U	C	Column Code	Sort
SALARY_HISTORY_PK	Yes	No	Yes	No	EMP_ID_CODE BEGIN_DATE	ASC ASC

Reference to List

Reference to	Primary Key	Foreign Key
EMPLOYEE	EMP_ID_CODE	EMP_ID_CODE

Table sponsor

Name:	sponsor
Code:	SPONSOR
Label:	Research Sponsor Info
Number:	
PK constraint:	

Options

Column List

Name	Code	Type	P	M
address	ADDRESS	char(40)	No	No
city	CITY	char(15)	No	No
fax	FAX	char(13)	No	No
name	NAME	char(30)	No	No
phone	PHONE	char(13)	No	No
spon_id_code	SPON_ID_CODE	char(6)	Yes	Yes
state	STATE	char(2)	No	No
zipcode	ZIPCODE	char(10)	No	No

ADDRESS

Check

Domain:
Low value:
High value:
Default value:
Unit:
Format:
Uppercase: No **Lowercase:** No **Can't modify:** No
List of values:

CITY

Check

Domain:
Low value:
High value:
Default value:

Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

FAX

Check

Domain:					
Low value:					
High value:					
Default value:					
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

NAME

Check

Domain:					
Low value:					
High value:					
Default value:					
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

PHONE

Check

Domain:					
Low value:					
High value:					
Default value:					
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

SPON_ID_CODE

Check

Domain:					
Low value:					
High value:					
Default value:					
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

STATE

Check

Domain:					
Low value:					
High value:					
Default value:					
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

ZIPCODE

Check

Domain:
Low value:
High value:
Default value:
Unit:
Format:
Uppercase: No **Lowercase:** No **Can't modify:** No
List of values:

Index List

Index Code	P	F	U	C	Column Code	Sort
SPONSOR_PK	Yes	No	Yes	No	SPON_ID_CODE	ASC

Reference by List

Referenced by	Primary Key	Foreign Key
ACCOUNT	SPON_ID_CODE	SPON_ID_CODE

Table staff

Name: staff
Code: STAFF
Label: Staff Specialization of Employee Table
Number:
PK constraint:

Options

Column List

Name	Code	Type	P	M
civ_grade	CIV_GRADE	char(5)	No	No
emp_id_code	EMP_ID_CODE	char(4)	Yes	Yes
step	STEP	char(2)	No	No

CIV_GRADE

Check

Domain:
Low value:
High value:
Default value:
Unit:
Format:
Uppercase: No **Lowercase:** No **Can't modify:** No
List of values:

EMP_ID_CODE

Check

Domain:
Low value:
High value:
Default value:
Unit:
Format:
Uppercase: No **Lowercase:** No **Can't modify:** No
List of values:

STEP

Check

Domain:

Low value:
High value:
Default value:
Unit:
Format:
Uppercase: No **Lowercase:** No **Can't modify:** No
List of values:

Index List

Index Code	P	F	U	C	Column Code	Sort
STAFF_PK	Yes	Yes	Yes	No	EMP_ID_CODE	ASC

Reference to List

Reference to	Primary Key	Foreign Key
EMPLOYEE	EMP_ID_CODE	EMP_ID_CODE

Table travel

Name: travel
Code: TRAVEL
Label: Travel Order Info
Number:
PK constraint:

Options

Column List

Name	Code	Type	P	M
actual_cost	ACTUAL_COST	decimal(10,2)	No	No
destination	DESTINATION	char(20)	No	No

fy_ending	FY_ENDING	date	No	Yes
jon	JON	char(5)	No	Yes
num_trav_days	NUM_TRAV_DAYS	integer	No	No
proj_cost	PROJ_COST	decimal(10,2)	No	No
to#	TO#	char(15)	Yes	Yes
to_date	TO_DATE	date	No	No
trav_start_date	TRAV_START_DATE	date	No	No

ACTUAL_COST

Check

Domain:
Low value: 0.00
High value:
Default value:
Unit:
Format:
Uppercase: No **Lowercase:** No **Can't modify:** No
List of values:

DESTINATION

Check

Domain:
Low value:
High value:
Default value:
Unit:
Format:
Uppercase: No **Lowercase:** No **Can't modify:** No
List of values:

FY_ENDING

Check

Domain:					
Low value:					
High value:					
Default value:	09/30/1997				
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

JON

Check

Domain:					
Low value:					
High value:					
Default value:					
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

NUM_TRAV_DAYS

Check

Domain:					
Low value:	1				
High value:					
Default value:					
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

PROJ_COST

Check

Domain:					
Low value:	0.00				
High value:					
Default value:					
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

TO#

Check

Domain:					
Low value:					
High value:					
Default value:					
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

TO_DATE

Check

Domain:	
Low value:	
High value:	
Default value:	
Unit:	

Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

TRAV_START_DATE

Check

Domain:					
Low value:					
High value:					
Default value:					
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

Index List

Index Code	P	F	U	C	Column Code	Sort
TRAVEL_FK1	No	Yes	No	No	JON	ASC
TRAVEL_PK	Yes	No	Yes	No	TO#	ASC

Reference to List

Reference to	Primary Key	Foreign Key
ACCOUNT	JON	JON

Reference by List

Referenced by	Primary Key	Foreign Key
TRAVEL_REQUESTS	TO#	TO#

Table travel_requests

Name:	travel_requests

Code:	TRAVEL_REQUESTS
Label:	Information on travelers for a specific Travel Order
Number:	
PK constraint:	

Options

Column List

Name	Code	Type	P	M
to#	TO#	char(15)	Yes	Yes
trav_fname	TRAV_FNAME	char(15)	Yes	Yes
trav_lname	TRAV_LNAME	char(15)	Yes	Yes
trav_mi	TRAV_MI	char(1)	No	No

TO#

Check

Domain:					
Low value:					
High value:					
Default value:					
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

TRAV_FNAME

Check

Domain:	
Low value:	
High value:	

Default value:					
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

TRAV_LNAME

Check

Domain:					
Low value:					
High value:					
Default value:					
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

TRAV_MI

Check

Domain:					
Low value:					
High value:					
Default value:					
Unit:					
Format:					
Uppercase:	No	**Lowercase:**	No	**Can't modify:**	No
List of values:					

Index List

Index Code	P	F	U	C	Column Code	Sort
TRAVEL_REQUESTS_PK	Yes	No	Yes	No	TO# TRAV_LNAME TRAV_FNAME	ASC ASC ASC

Reference to List

Reference to	Primary Key	Foreign Key
TRAVEL	TO#	TO#

Views

View List

Name	Code	Upd	Gen
dr_chgs	DR_CHGS	No	Yes
dt_chgs	DT_CHGS	No	Yes
rr_chgs	RR_CHGS	No	Yes
st_fmt_chgs	ST_FMT_CHGS	Yes	Yes
st_ind_chgs	ST_IND_CHGS	Yes	Yes
st_ir_chgs	ST_IR_CHGS	Yes	Yes
st_omn_chgs	ST_OMN_CHGS	Yes	Yes
st_ot_chgs	ST_OT_CHGS	Yes	Yes
st_rr_chgs	ST_RR_CHGS	Yes	Yes
st_tuit_chgs	ST_TUIT_CHGS	Yes	Yes

View dr_chgs

Name: dr_chgs
Code: DR_CHGS
Label: DR Charges View
Usage: Query Only
Generate View

Code

```
select FACULTY.EMP_ID_CODE, LABOR_CHGS.PPE_DATE, LABOR_CHGS.HOURS
from ACCOUNT, FACULTY, LABOR_CHGS
where FACULTY.EMP_ID_CODE = LABOR_CHGS.EMP_ID_CODE
and ACCOUNT.JON = LABOR_CHGS.JON
and ACCOUNT.FUND_TYPE = 'DR'
```

View dt_chgs

Name: dt_chgs
Code: DT_CHGS
Label: DT Charges View
Usage: Query Only
Generate View

Code

```
select FACULTY.EMP_ID_CODE, LABOR_CHGS.PPE_DATE, LABOR_CHGS.HOURS
from ACCOUNT, FACULTY, LABOR_CHGS
where FACULTY.EMP_ID_CODE = LABOR_CHGS.EMP_ID_CODE
and ACCOUNT.JON = LABOR_CHGS.JON
and ACCOUNT.FUND_TYPE = 'DT'
```

View rr_chgs

Name: rr_chgs
Code: RR_CHGS
Label: RR Charges View
Usage: Query Only
Generate View

Code

```
select FACULTY.EMP_ID_CODE, ACCOUNT.LABOR_JON, LABOR_CHGS.PPE_DATE, LABOR_CHGS.HOURS
from ACCOUNT, FACULTY, LABOR_CHGS
where FACULTY.EMP_ID_CODE = LABOR_CHGS.EMP_ID_CODE
and ACCOUNT.JON = LABOR_CHGS.JON
and ACCOUNT.FUND_TYPE = 'RR'
```

View st_fmt_chgs

Name:	st_fmt_chgs
Code:	ST_FMT_CHGS
Label:	st_fmt_chgs
Usage:	Updatable
	Generate View
	With check option

Code

```
select STAFF.EMP_ID_CODE, LABOR_CHGS.HOURS, LABOR_CHGS.PPE_DATE
from STAFF, LABOR_CHGS
where STAFF.EMP_ID_CODE = LABOR_CHGS.EMP_ID_CODE
and LABOR_CHGS.JON = 'FMT'
```

View st_ind_chgs

Name:	st_ind_chgs
Code:	ST_IND_CHGS
Label:	st_ind_chgs
Usage:	Updatable
	Generate View
	With check option

Code

```
select STAFF.EMP_ID_CODE, LABOR_CHGS.HOURS, LABOR_CHGS.PPE_DATE
from STAFF, LABOR_CHGS
where STAFF.EMP_ID_CODE = LABOR_CHGS.EMP_ID_CODE
and LABOR_CHGS.JON = 'IND'
```

View st_ir_chgs

Name:	st_ir_chgs
Code:	ST_IR_CHGS
Label:	st_ir_chgs
Usage:	Updatable

Generate View With check option

Code

```
select ACCOUNT.LABOR_JON, STAFF.EMP_ID_CODE, LABOR_CHGS.HOURS, LABOR_CHGS.PPE_DATE
from ACCOUNT, STAFF, LABOR_CHGS
where STAFF.EMP_ID_CODE = LABOR_CHGS.EMP_ID_CODE
and ACCOUNT.JON = LABOR_CHGS.JON
and ACCOUNT.FUND_TYPE = 'IR'
```

View st_omn_chgs

Name:	st_omn_chgs
Code:	ST_OMN_CHGS
Label:	st_omn_chgs
Usage:	Updatable Generate View With check option

Code

```
select STAFF.EMP_ID_CODE, LABOR_CHGS.HOURS, LABOR_CHGS.PPE_DATE
from STAFF, LABOR_CHGS
where STAFF.EMP_ID_CODE = LABOR_CHGS.EMP_ID_CODE
and LABOR_CHGS.JON = 'O&MN'
```

View st_ot_chgs

Name:	st_ot_chgs
Code:	ST_OT_CHGS
Label:	st_ot_chgs
Usage:	Updatable Generate View With check option

Code

```
select ACCOUNT.LABOR_JON, STAFF.EMP_ID_CODE, LABOR_CHGS.OT_HOURS, LABOR_CHGS.PPE_DATE
```

from ACCOUNT, STAFF, LABOR_CHGS
where STAFF.EMP_ID_CODE = LABOR_CHGS.EMP_ID_CODE
and ACCOUNT.JON = LABOR_CHGS.JON
and LABOR_CHGS.OT_HOURS > 0

View st_rr_chgs

Name:	st_rr_chgs
Code:	ST_RR_CHGS
Label:	st_rr_chgs
Usage:	Updatable
	Generate View
	With check option

Code

select ACCOUNT.LABOR_JON, STAFF.EMP_ID_CODE, LABOR_CHGS.HOURS, LABOR_CHGS.PPE_DATE
from ACCOUNT, STAFF, LABOR_CHGS
where STAFF.EMP_ID_CODE = LABOR_CHGS.EMP_ID_CODE
and ACCOUNT.JON = LABOR_CHGS.JON
and ACCOUNT.FUND_TYPE = 'RR'

View st_tuit_chgs

Name:	st_tuit_chgs
Code:	ST_TUIT_CHGS
Label:	st_tuit_chgs
Usage:	Updatable
	Generate View
	With check option

Code

select STAFF.EMP_ID_CODE, LABOR_CHGS.HOURS, LABOR_CHGS.PPE_DATE
from STAFF, LABOR_CHGS
where STAFF.EMP_ID_CODE = LABOR_CHGS.EMP_ID_CODE
and LABOR_CHGS.JON = 'TUIT'

Triggers

Trigger List

Table	Trigger	User-Defined
ACCOUNT	tib_account	No
ACCOUNT	tia_account	Yes
ACCOUNT	tua_account	Yes
ADP_PROJ_INFO	tib_adp_proj_info	No
CONTRACTS	tib_contracts	No
CONTRACTS	tia_contracts	Yes
CONTRACTS	tub_contracts	No
CONTRACTS	tua_contracts	Yes
CONTRACTS	tda_contracts	Yes
EMPLOYEE	tib_employee	No
FACULTY	tib_faculty	No
LABOR_CHGS	tib_labor_chgs	No
LABOR_CHGS	tia_labor_chgs	Yes
LABOR_CHGS	tua_labor_chgs	Yes
LABOR_CHGS	tda_labor_chgs	Yes
LABOR_LES	tib_labor_les	No
MILITARY	tib_military	No
OPTAR_REQ	tib_optar_req	No
OPTAR_REQ	tia_optar_req	Yes
OPTAR_REQ	tua_optar_req	Yes
OPTAR_REQ	tda_optar_req	Yes
OTHER_LEAVE	tib_other_leave	No
OTHER_LEAVE	tub_other_leave	No
PI	tib_pi	No
SALARY_HISTORY	tib_salary_history	No
STAFF	tib_staff	No
TRAVEL	tib_travel	No
TRAVEL	tia_travel	Yes
TRAVEL	tua_travel	Yes
TRAVEL	tda_travel	Yes
TRAVEL_REQUESTS	tib_travel_requests	No

Procedures

Procedure List

Name	Code	Func
calc_bal_contract	CALC_BAL_CONTRACT	No
calc_bal_fac_labor	CALC_BAL_FAC_LABOR	No
calc_bal_optar	CALC_BAL_OPTAR	No
calc_bal_spt_labor	CALC_BAL_SPT_LABOR	No
calc_bal_trav	CALC_BAL_TRAV	No

INITIAL DISTRIBUTION LIST

No. Copies

1. Defense Technical Information Center 2
   8725 John J. Kingman Rd., STE 0944
   Ft. Belvoir, VA 22060-6218

2. Dudley Knox Library 2
   Naval Postgraduate School
   411 Dyer Rd.
   Monterey, CA 93943-5101

3. Chairman, Computer Science Department 1
   Code CS/Lt
   Naval Postgraduate School
   Monterey, CA 93943-5000

4. Chairman, Operations Research Department 1
   Code OR/Pe
   Naval Postgraduate School
   Monterey, CA 93943-5000

5. Dr. C. Thomas Wu 1
   Code CS/Wu
   Computer Science Department
   Naval Postgraduate School
   Monterey, CA 93943-5000

6. Dr. James Emery 1
   Code 05
   Associate Provost for Computer and Information Services
   Naval Postgraduate School
   Monterey, CA 93943-5000

7. LCDR John A. Daley 1
   Code CS/Da
   Computer Science Department
   Naval Postgraduate School
   Monterey, CA 93943-5000

8. Alan E. Pires 2
Code OR
Operations Research Department
Naval Postgraduate School
Monterey, CA 93943-5000

9. Mr. and Mrs. Robert D. Pires 1
920-35th Street
Richmond, CA 94805